Data Mining Project Phase 1

Stroke Prediction

Team Members :

Ali Ashraf Hamdy – 23P0233
Omar Fouad Abdelhamid – 23P0146
Seif El-Din Tamer Safwat – 23P0240
Asaad Ramzy Asaad – 23P0228
Adel Waheed Mohamed – 23P0213
Kirollous Ramzy – 22P0194
Ahmed Tarek – 22P0301

AIN SHAMS UNIVERISTY

Introduction

Stroke is one of the leading causes of death and long-term disability worldwide. Early detection of individuals at high risk of experiencing a stroke can significantly improve prevention and treatment outcomes. With the rise of healthcare data and machine learning, predictive models have become powerful tools for identifying key risk factors and forecasting potential health issues.

This project aims to **analyze patient health records**, perform **data preprocessing**, and build **machine learning models** capable of predicting the likelihood of a stroke. The project also explores data visualization, outlier detection, feature engineering, and classification algorithms to understand both the structure of the data and the patterns behind stroke occurrences.

Dataset Description

The dataset used in this project is the **Stroke Prediction Dataset**, which contains various health-related attributes for a group of individuals. Each record represents a single person, with demographic information, lifestyle details, and medical measurements. The dataset includes the following features:

◇ Demographic Features

- **gender** – Male or Female.
- **age** – Age of the individual.
- **ever_married** – Whether the individual has ever been married.
- **Residence_type** – Rural or Urban residency.

◇ Lifestyle and Social Factors

- **work_type** – Type of occupation (Private, Self-employed, Govt job, Children, Never worked).
- **smoking_status** – Current smoking habits (formerly smoked, never smoked, smokes, unknown).

◇ Medical Attributes

- **hypertension** – Indicates if the person has hypertension (1 = yes, 0 = no).

- **heart_disease** – Indicates if the person has a heart condition (1 = yes, 0 = no).
- **avg_glucose_level** – Average blood glucose level.
- **bmi** – Body Mass Index.

◇ Target Variable

- **stroke** – Indicates whether the individual had a stroke (1 = yes, 0 = no).
  This is a **binary classification** target.

Dataset types

```
id                                int64
gender                            int32
age                             float64
hypertension                      int64
heart_disease                     int64
ever_married                      int32
Residence_type                    int32
avg_glucose_level               float64
bmi                             float64
stroke                            int64
avg_glucose_level_log           float64
work_type_Never_worked             bool
work_type_Private                  bool
work_type_Self-employed            bool
work_type_children                 bool
smoking_status_formerly smoked     bool
smoking_status_never smoked        bool

smoking_status_smokes              bool
```

Data Understanding Phase

This phase focuses on **loading the dataset**, **examining its structure**, and gaining an initial idea of its contents. The code contributes to this phase in the following way:

1. Loading the Dataset

The `pd.read_csv()` function brings the dataset into the environment so it can be inspected and analyzed.
This is the first step in understanding what data you are working with.

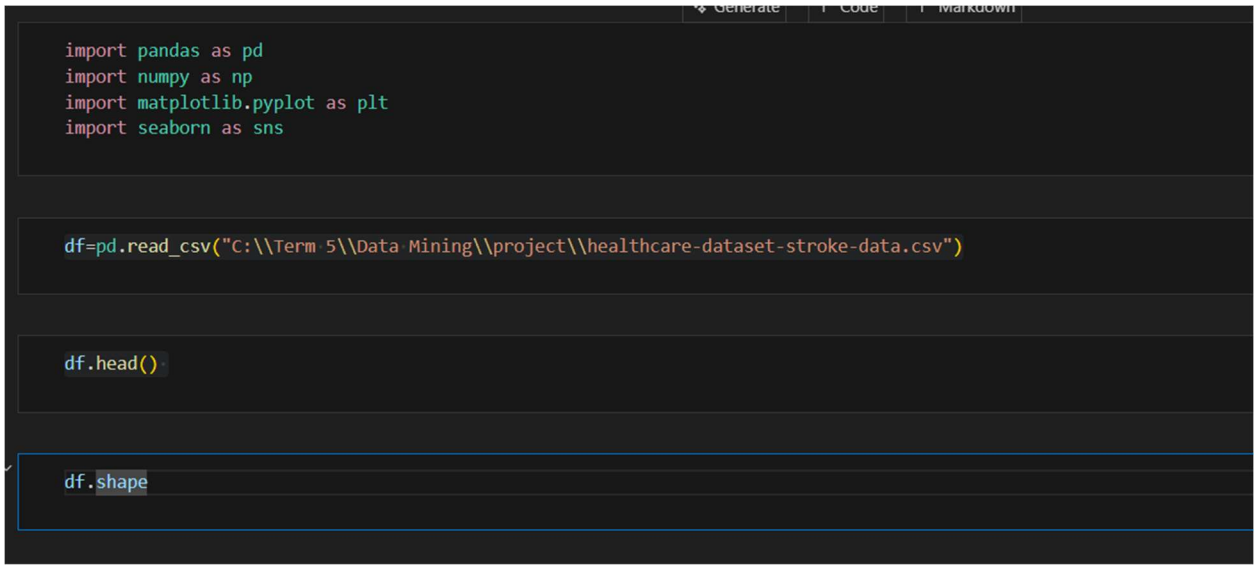
2. Inspecting Dataset Size

The command `df.shape` provides the dimensions of the dataset:

- Number of rows (records or observations)
- Number of columns (features or attributes)

Knowing the dataset size is essential to:

- Understand the scale of the problem
- Prepare for cleaning, preprocessing, and modeling
- Identify if the dataset is large, small, or requires adjustments



```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

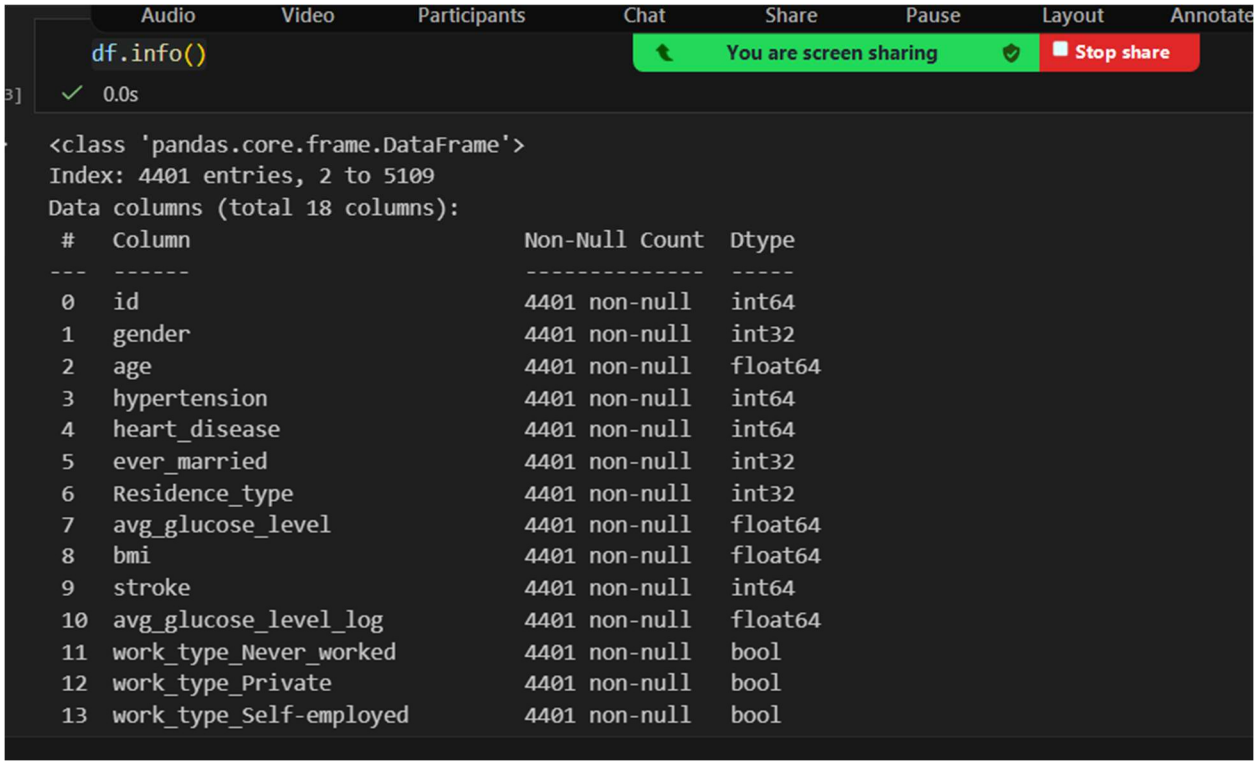
```
df=pd.read_csv("C:\\Term 5\\Data Mining\\project\\healthcare-dataset-stroke-data.csv")
```

```
df.head()
```

```
df.shape
```

```
our dataset contain 12  columns and 5110 rows

# features types
## numeric
age
```

```
avg_glucose_level

bmi
## boolean and categorical

gender

hypertension

heart_disease

ever_married

work_type

Residence_typr

smoking_status

stroke
```

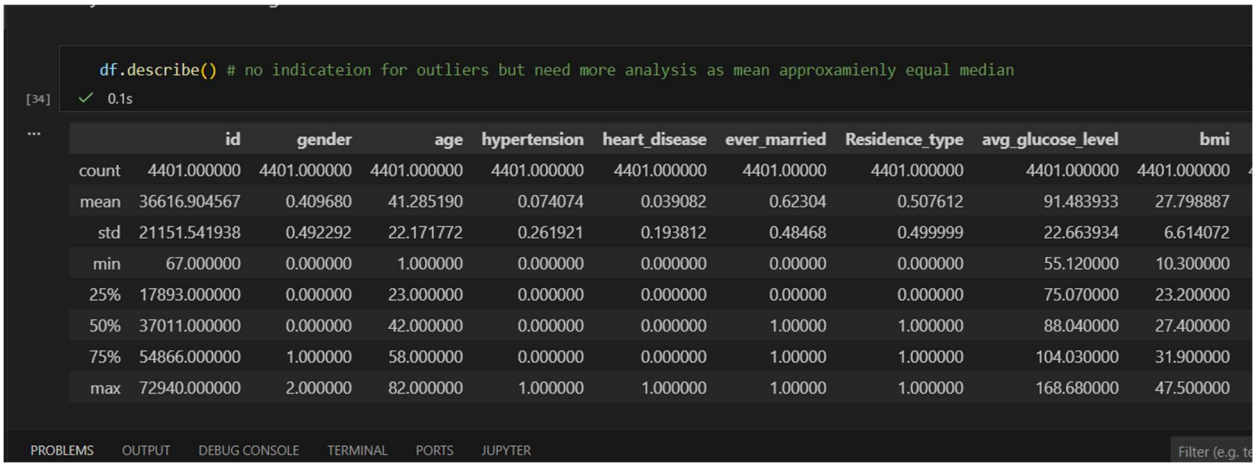


`df.info()`

This command displays a summary of the dataset, including:

- Column names
- Number of non-null (non-missing) values
- Data types of each column
- Total number of entries

Purpose:
It helps identify which columns contain missing values and what data types (e.g., integers, floats, objects) need to be handled during preprocessing.



```
df.describe() # no indicateion for outliers but need more analysis as mean approxamienly equal median
```

[34] ✓ 0.1s

	id	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi
count	4401.000000	4401.000000	4401.000000	4401.000000	4401.000000	4401.00000	4401.000000	4401.000000	4401.000000
mean	36616.904567	0.409680	41.285190	0.074074	0.039082	0.62304	0.507612	91.483933	27.798887
std	21151.541938	0.492292	22.171772	0.261921	0.193812	0.48468	0.499999	22.663934	6.614072
min	67.000000	0.000000	1.000000	0.000000	0.000000	0.00000	0.000000	55.120000	10.300000
25%	17893.000000	0.000000	23.000000	0.000000	0.000000	0.00000	0.000000	75.070000	23.200000
50%	37011.000000	0.000000	42.000000	0.000000	0.000000	1.00000	1.000000	88.040000	27.400000
75%	54866.000000	1.000000	58.000000	0.000000	0.000000	1.00000	1.000000	104.030000	31.900000
max	72940.000000	2.000000	82.000000	1.000000	1.000000	1.00000	1.000000	168.680000	47.500000

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER Filter (e.g. te

`df.describe()`

This function generates descriptive statistics for **numerical** columns, such as:

- Count
- Mean
- Standard deviation
- Minimum and maximum values
- Quartiles (25%, 50%, 75%)

Purpose:
It provides an initial understanding of the distribution of numerical data, detects unusual values, and highlights potential outliers.

```
for column in df.select_dtypes(include=['object']).columns:
    print(f"{column}: {df[column].nunique()} unique values")
```

3. Counting Unique Values in Categorical Columns

Explanation:

- `df.select_dtypes(include=['object'])` selects all categorical columns.
- `df[column].nunique()` counts how many distinct values each categorical column contains.
- The loop prints the number of unique categories for each categorical feature.

Purpose:
This step helps determine:

- How many categories each feature has
- Whether a column has too many or too few categories
- What type of encoding (Label Encoding or One-Hot Encoding) is appropria

`df.duplicated()`

- This function checks each row in the dataset and determines whether it is a **duplicate** of a previous row.
- It returns a series of **True/False** values:
  - **True** → the row is duplicated
  - **False** → the row is unique

`.sum()`

- Since **True = 1** and **False = 0**, summing the results gives the **total number of duplicated rows** in the dataset.

📌 Target Variable Analysis

Understanding the distribution of the target variable (**stroke**) is an essential step before building machine learning models. Since this is a **classification problem**, analyzing the target helps identify whether the dataset is balanced or imbalanced, which affects the choice of evaluation metrics and preprocessing techniques such as SMOTE or class weighting.

The following code visualizes the distribution of the target variable:

```
plt.figure(figsize=(5,4))
sns.countplot(x="stroke", data=df)
plt.title("Distribution of Stroke Cases")
plt.xlabel("Stroke (0 = No, 1 = Yes)")
plt.ylabel("Count")
plt.show()
#most cases doesnt have stroke
```

1. `plt.figure(figsize=(5,4))`

Creates a new figure for the plot with a defined width and height.
This ensures the visualization is displayed clearly and proportionally.

2. `sns.countplot(x="stroke", data=df)`

This line creates a **count plot** using Seaborn.
A count plot shows how many times each category appears.

- `x="stroke"` → The target variable on the x-axis
- `data=df` → The dataset to use

This allows us to see how many individuals had a stroke (1) versus those who did not (0).

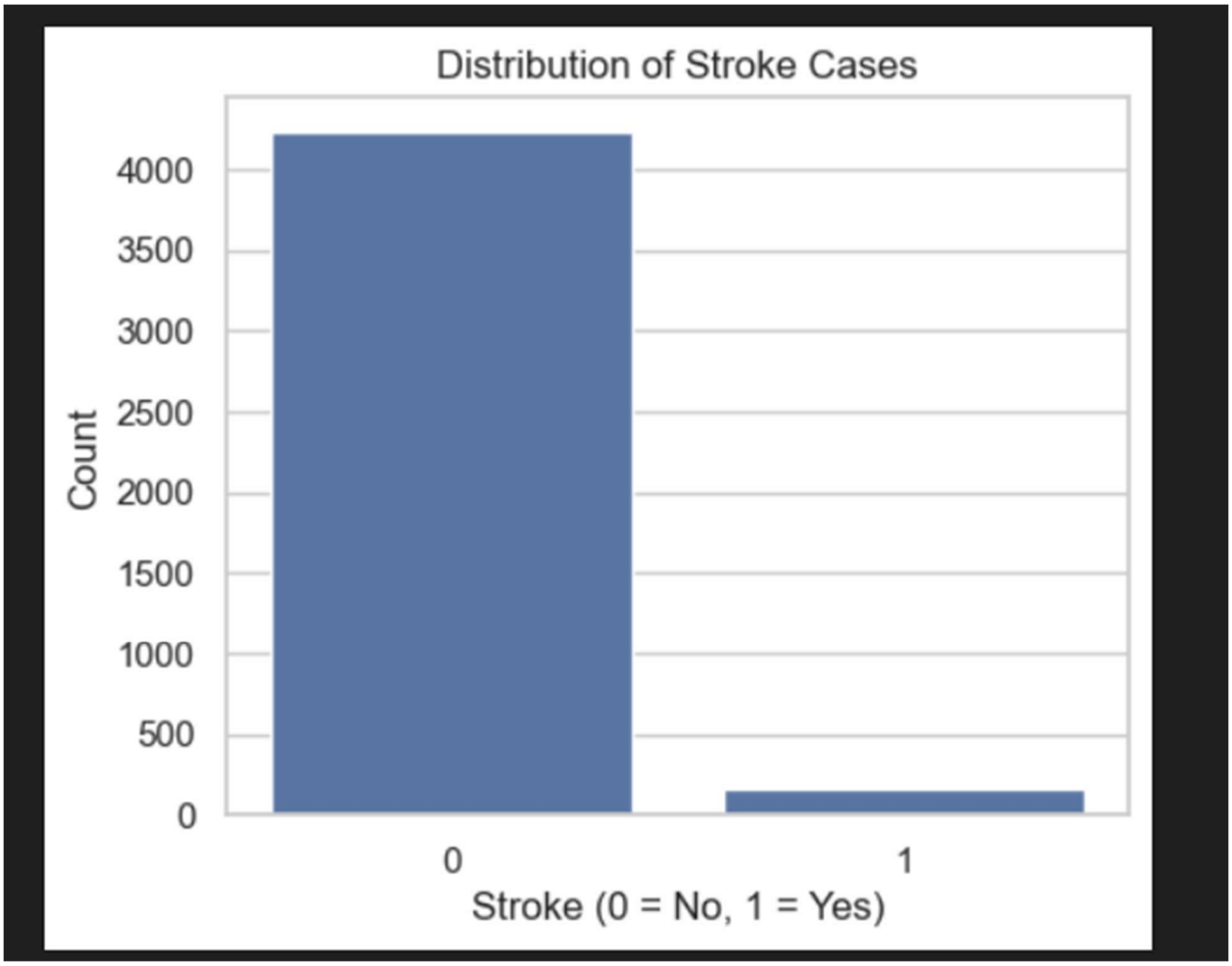
3. Labels and Title

- `plt.title()` → Adds a title to the chart
- `plt.xlabel()` → Labels the x-axis
- `plt.ylabel()` → Labels the y-axis

These labels make the plot easy to interpret.

4. `plt.show()`

Displays the final plot.



`df['stroke']`

This selects the **stroke** column from the dataset.
This column contains the target variable with values:

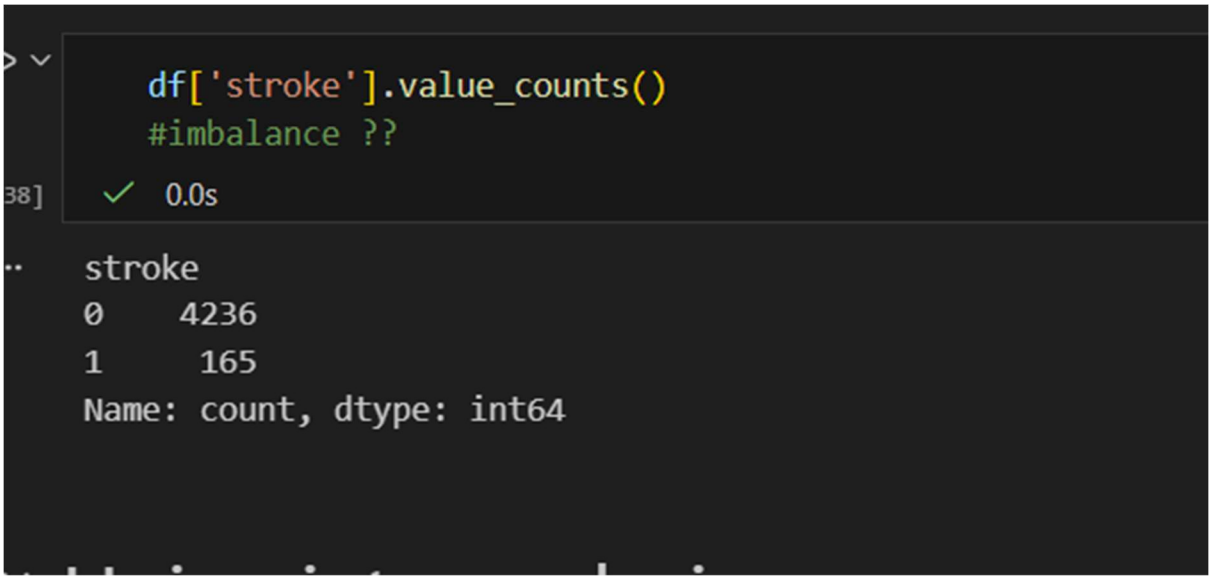
- **0** → No stroke
- **1** → Stroke

`value_counts()`

This function counts how many times each unique value appears in the selected column.

It returns:

- The number of people who **did not** have a stroke (0)
- The number of people who **did** have a stroke (1)



📌 Univariate Analysis

Univariate analysis focuses on examining each variable individually to understand its distribution, structure, and statistical characteristics. This step is essential in the **Data Understanding** phase of the data mining process, as it helps identify patterns, detect outliers, and understand the nature of both numerical and categorical features before applying further preprocessing or modeling.

Below is the full analysis for both **numerical** and **categorical** features.

🔹 1. Univariate Analysis for Numerical Features

```
num_features = ['age', 'bmi', 'avg_glucose_level']

# Set visual style
sns.set(style="whitegrid")

for col in num_features:
    plt.figure(figsize=(6,4))
    sns.histplot(df[col], kde=True, bins=30, color='skyblue',
edgecolor='black')
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.show()
```

📌 Code Explanation

1. Selecting numerical features

```
num_features = ['age', 'bmi', 'avg_glucose_level']
```

These are the key numerical variables in the dataset. Each will be analyzed separately.

2. Setting visualization style

```
sns.set(style="whitegrid")
```

This improves the appearance of the plots for readability.

3. Looping through each numerical feature

The loop creates a separate **histogram** for each numerical column:

- `sns.histplot()` → Plots the distribution of the variable
- `kde=True` → Adds a smooth probability density curve
- `bins=30` → Controls the granularity of the histogram
- `edgecolor='black'` → Makes bars clearer

📌 Purpose of This Analysis

- Understand the distribution (normal, skewed, uniform)
- Identify outliers or extreme values
- Check for potential transformations (log scaling)
- Detect imbalanced ranges in variables such as glucose or BMI

🔹 2. Univariate Analysis for Categorical Features (Tabular Summary)

```
cat_features = [
    'gender',
    'hypertension',
    'heart_disease',
    'ever_married',
    'work_type',
    'Residence_type',
    'smoking_status',
    'stroke'  # include target for quick overview
]

for col in cat_features:
    print(f"\n===== {col.upper()} =====")
    counts = df[col].value_counts()
    percentages = df[col].value_counts(normalize=True) * 100
    summary = pd.DataFrame({'Count': counts, 'Percentage':
percentages.round(2)})
    print(summary)
```

📌 Code Explanation

1. Selecting categorical features

A list of all categorical variables, including the target variable.

2. `value_counts()`

Counts how many times each category appears.

3. `value_counts(normalize=True)`

Computes percentages instead of counts.

4. Creating a summary table

A DataFrame is created showing:

- **Count** of each category
- **Percentage** representation

📌 Purpose of This Analysis

- Understand category frequency distribution
- Identify rare categories
- Detect imbalanced categorical variables
- Inform decisions about encoding (One-Hot, Label Encoding)

🔹 3. Visualizing Categorical Features (Count Plots)

A. First Group of Categorical Features

```
cat_features_1 = ['gender', 'hypertension', 'heart_disease']

for col in cat_features_1:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df, palette='Set2', edgecolor='black',
hue=None, legend=False)
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=30)
    plt.show()
```

📌 Explanation

Each feature in this list is plotted using a **count plot**, which visualizes how many times each category appears. This helps detect:

- Imbalances (e.g., most people have no hypertension)
- Distribution patterns
- Whether categories are evenly spread

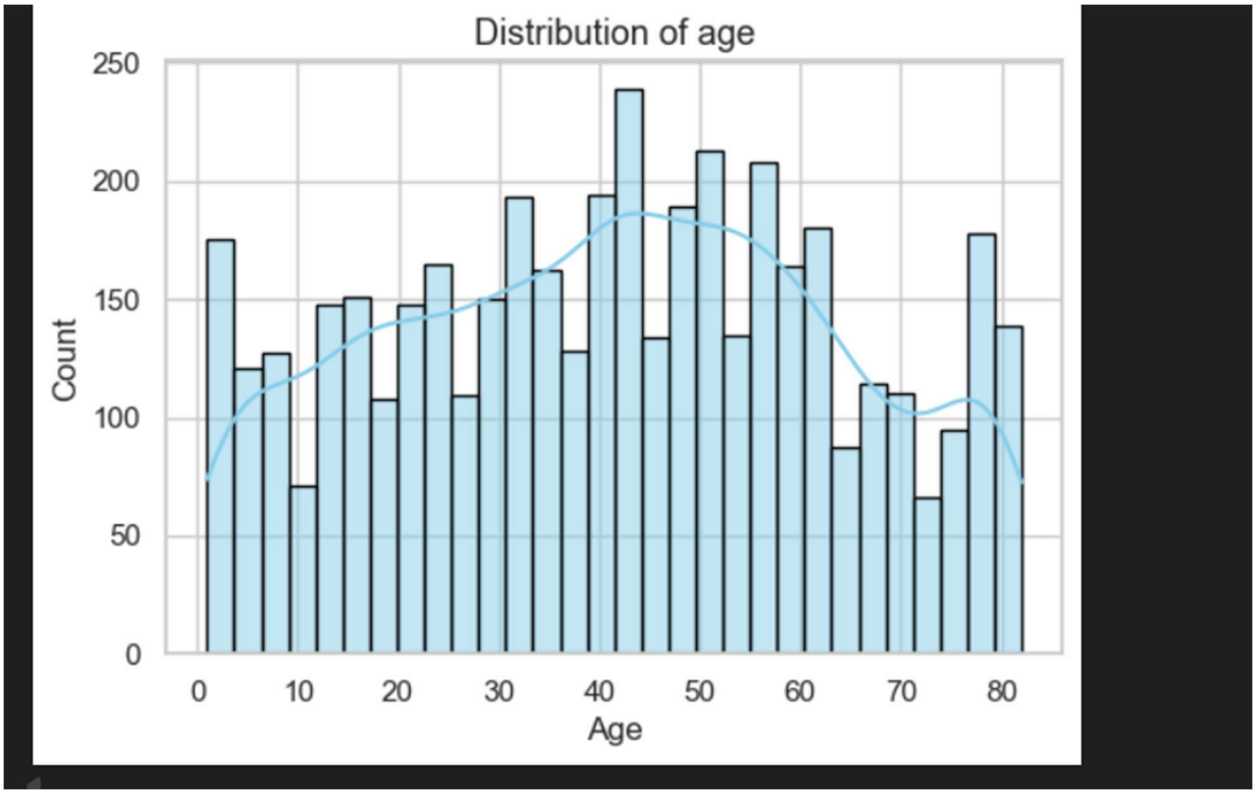
B. Second Group of Categorical Features

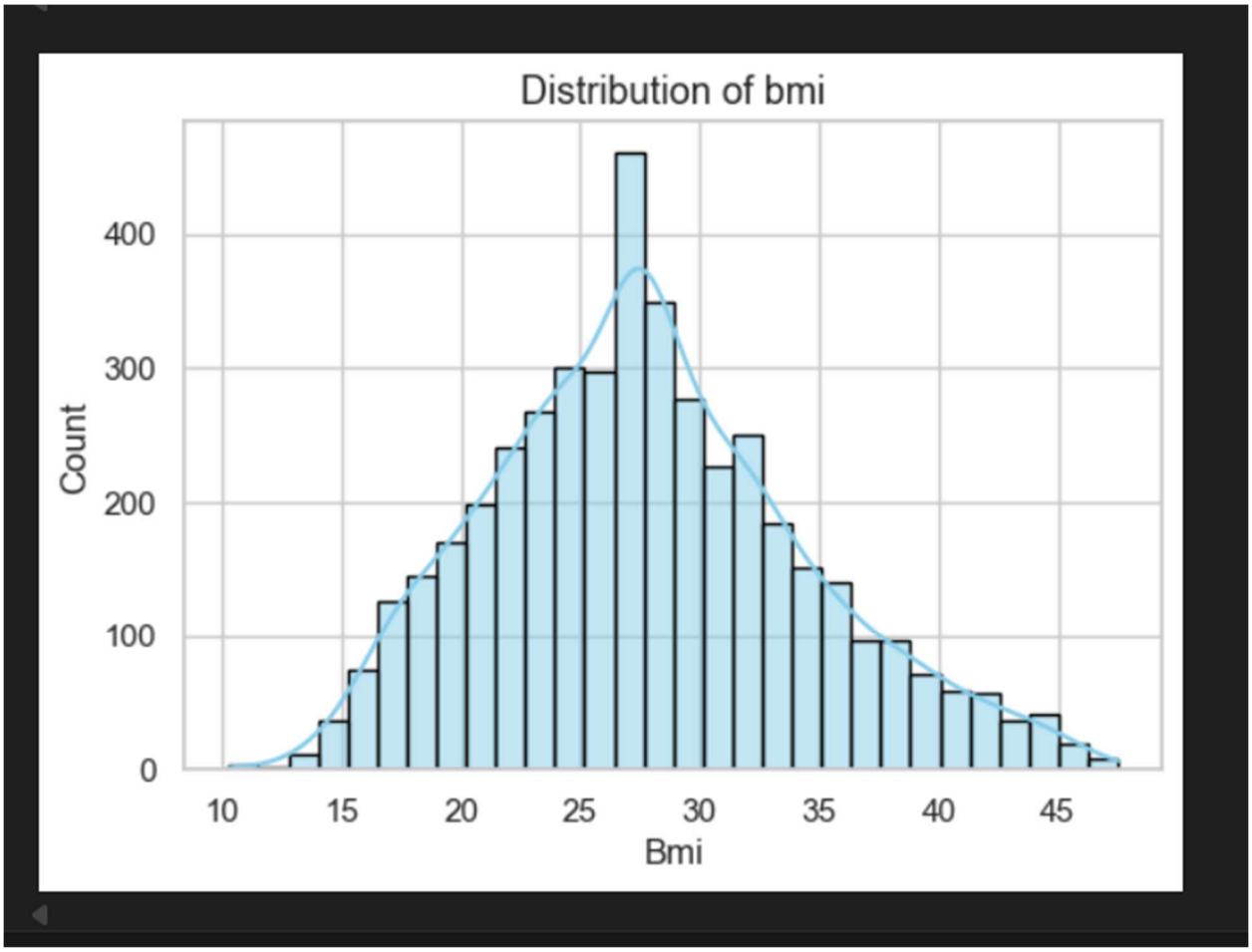
```
cat_features_2 = ['ever_married', 'work_type']

for col in cat_features_2:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df, palette='Set2', edgecolor='black',
hue=None, legend=False)
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=30)
    plt.show()
```

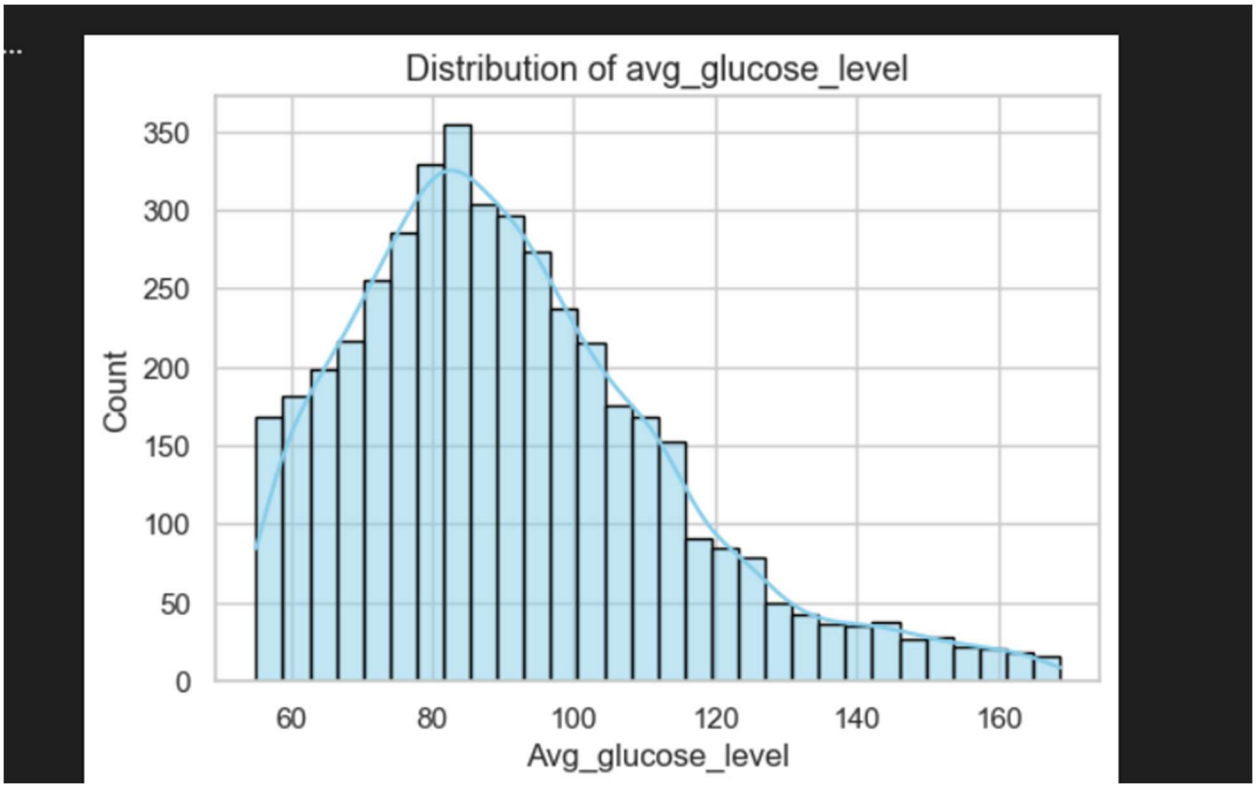
📌 Explanation

Same approach as the previous block, but for different features.
Rotating the x-axis labels improves readability, especially for `work_type`.

```
num_features = ['age', 'bmi', 'avg_glucose_level']

# Set visual style
sns.set(style="whitegrid")

for col in num_features:
    plt.figure(figsize=(6,4))
    sns.histplot(df[col], kde=True, bins=30, color='skyblue', edgecolor='black')
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.show()
```


Distribution of age
Count
250
200
150
100
50
0
0
10
20
30
40
50
60
70
80
Age

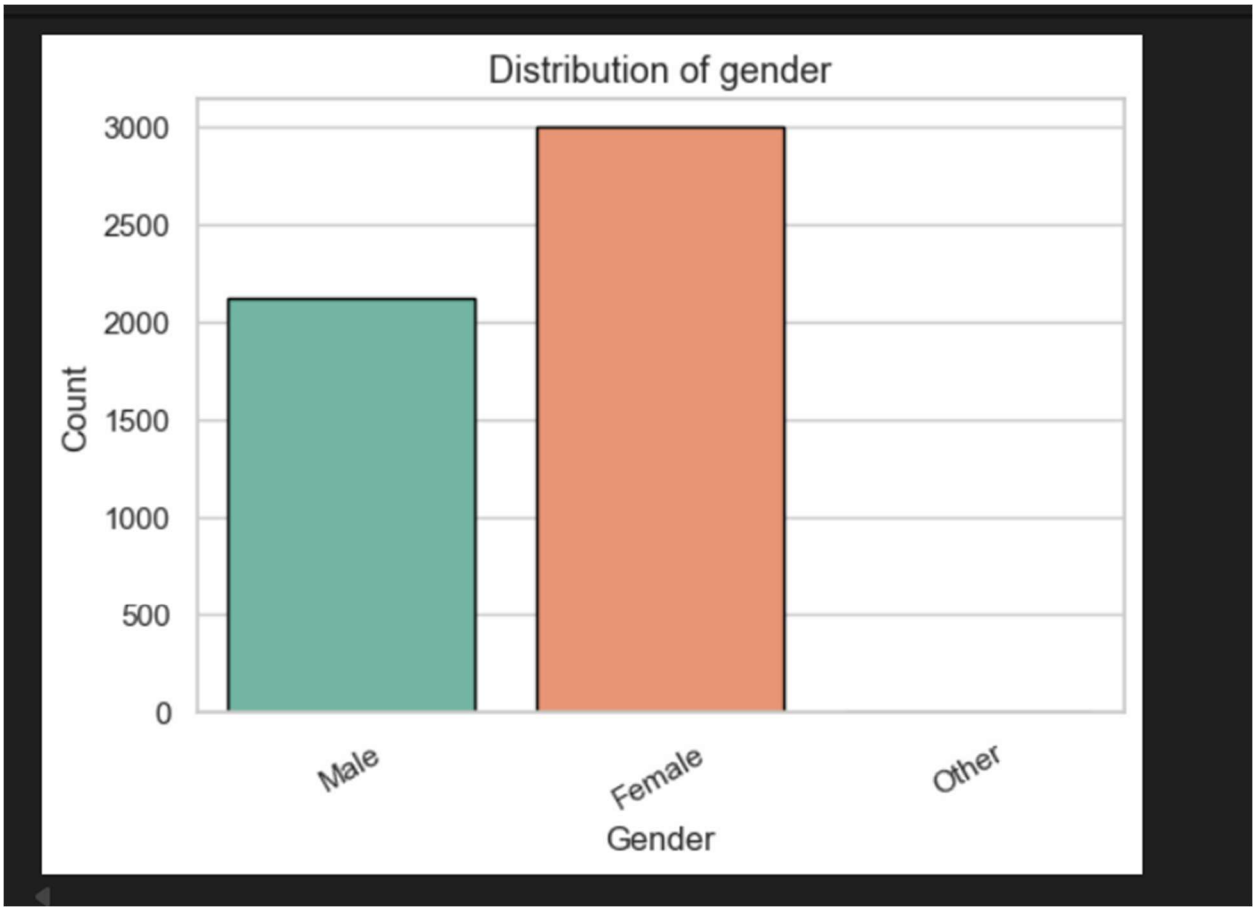

Distribution of bmi
Count
400
300
200
100
0
10
15
20
25
30
35
40
45
Bmi

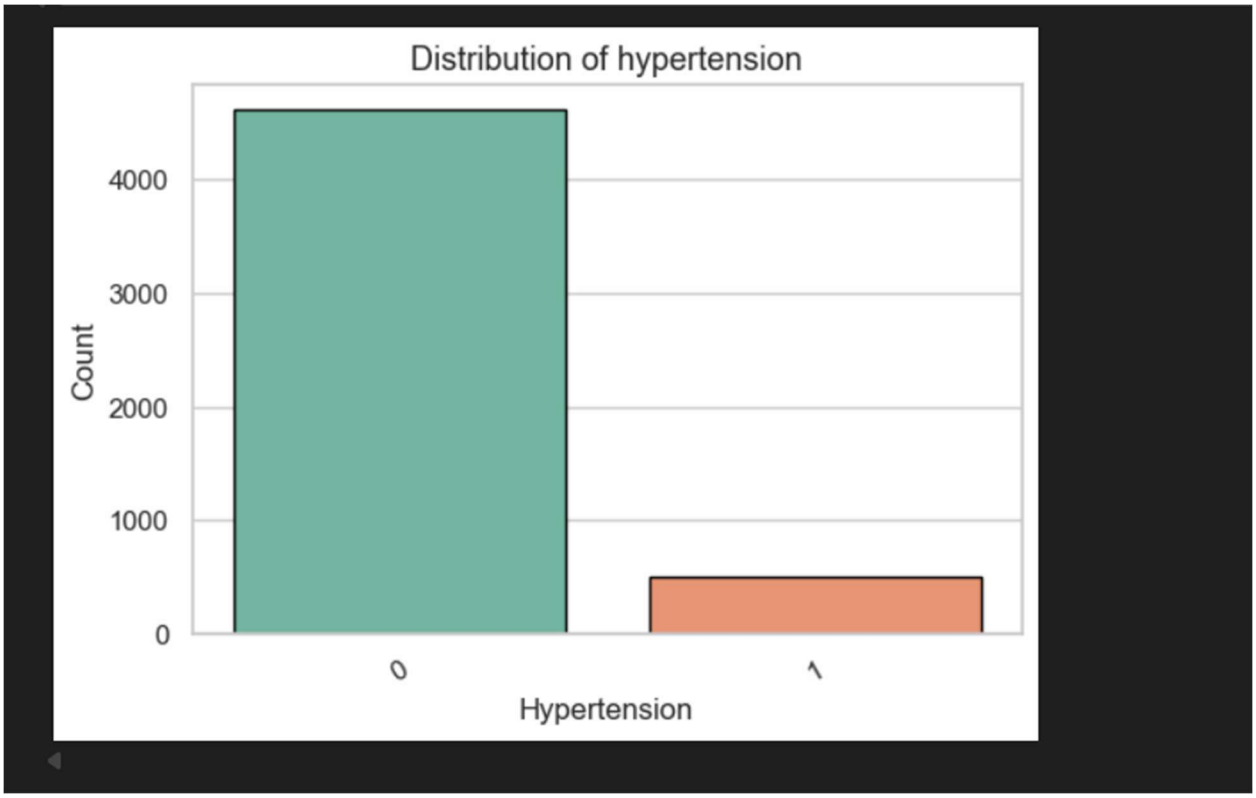


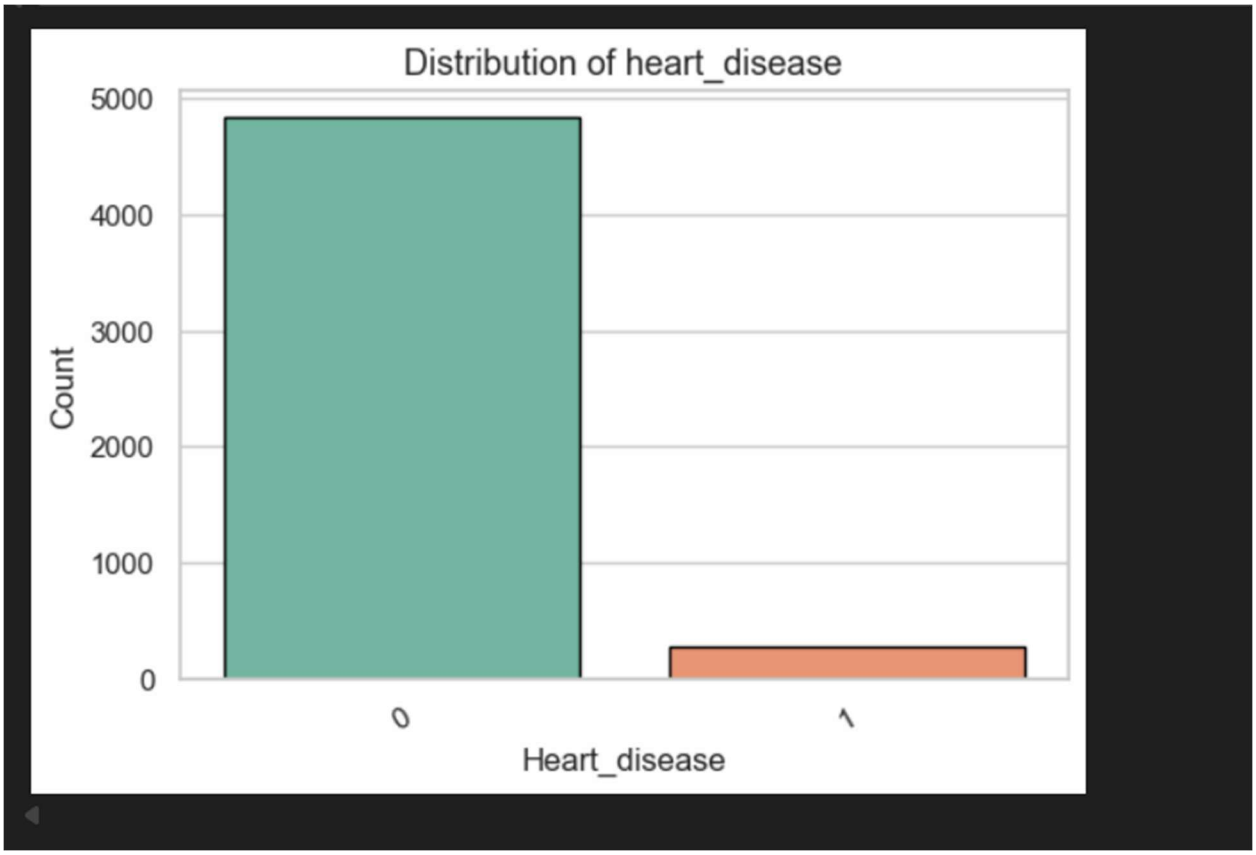

```
for col in cat_features:
    print(f"\n===== {col.upper()} =====")
    counts = df[col].value_counts()
    percentages = df[col].value_counts(normalize=True) * 100
    summary = pd.DataFrame({'Count': counts, 'Percentage': percentages.round(2)})
    print(summary)
```

```
===== GENDER =====
        Count  Percentage
gender
Female   2994       58.59
Male     2115       41.39
Other       1        0.02

===== HYPERTENSION =====
              Count  Percentage
hypertension
0              4612       90.25
1               498        9.75

===== HEART_DISEASE =====
               Count  Percentage
heart_disease
0               4834        94.6
1                276         5.4

===== EVER_MARRIED =====
              Count  Percentage
ever_married
Yes            3353       65.62
No             1757       34.38
```

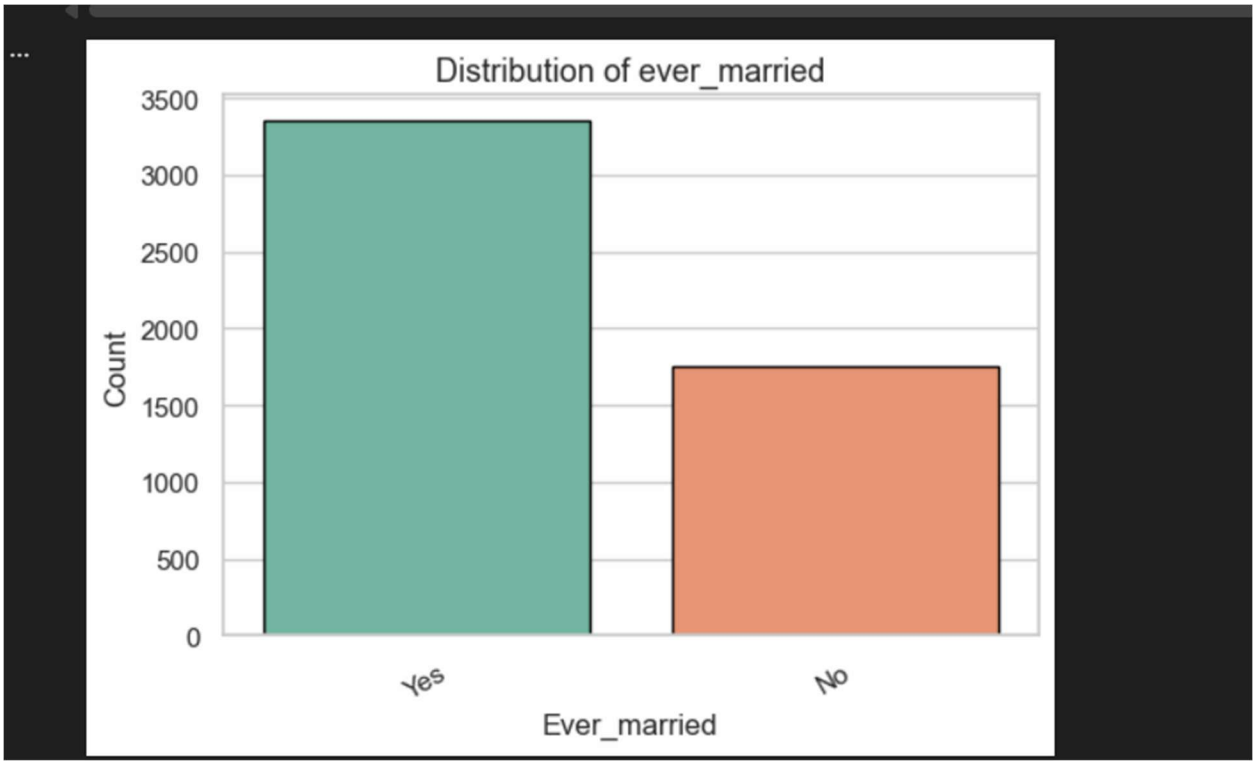
```
cat_features_1 = ['gender', 'hypertension', 'heart_disease']

for col in cat_features_1:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df, palette='Set2', edgecolor='black', hue=None,
legend=False)
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=30)
    plt.show()
```


Distribution of gender
3000
2500
2000
1500
1000
500
0
Count
Male
Female
Other
Gender


Distribution of hypertension
4000
3000
2000
1000
0
Count
0
1
Hypertension




```
cat_features_2 = ['ever_married', 'work_type']

for col in cat_features_2:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df, palette='Set2', edgecolor='black', hue=None,
legend=False)
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=30)
    plt.show()
```



1. Categorical Features Distribution

```
cat_features_3 = ['Residence_type', 'smoking_status', 'stroke']

for col in cat_features_3:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df, palette='Set2', edgecolor='black',
hue=None, legend=False)
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=30)
    plt.show()
```

Explanation

- **Purpose:** Visualize the distribution of key categorical variables including `Residence_type`, `smoking_status`, and the target `stroke`.
- **`sns.countplot()`**: Plots the frequency of each category.
- **Observations:**
  - Helps identify category imbalances.
  - Shows which categories dominate among stroke and non-stroke cases.

◇ 2. Numerical Features vs Target Variable

```
num_features = ['age', 'bmi', 'avg_glucose_level']

for col in num_features:
    plt.figure(figsize=(6,4))
```

```
    sns.boxplot(x='stroke', y=col, data=df, color='skyblue')
    plt.title(f'{col} vs Stroke')
    plt.xlabel('Stroke')
    plt.ylabel(col.capitalize())
    plt.show()
```

Explanation

- **Purpose:** Examine how numerical features vary across the target classes (`stroke = 0` or `1`).
- **`sns.boxplot()`**:
  - Visualizes distribution, median, quartiles, and outliers.
  - `x='stroke'` separates the data by target class.
  - `y=col` plots the numerical variable values.
- **Observations:**
  - Identify if people who experienced a stroke differ significantly in age, BMI, or glucose level compared to those who did not.
  - Outliers can be detected for potential treatment.

◇ 3. Mean Comparison Table

```
print(df.groupby('stroke')[num_features].mean())
```

Explanation

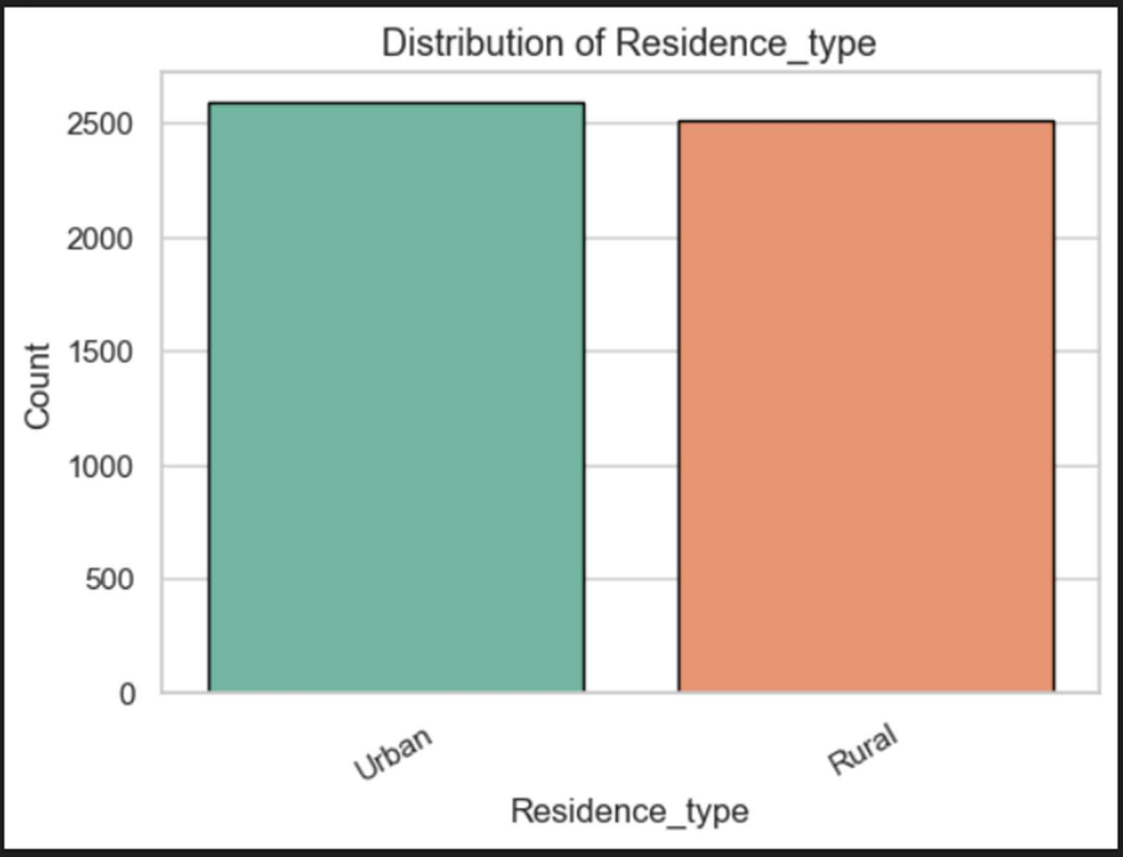
- **`df.groupby('stroke')`**: Groups the dataset by the target variable (`stroke`).
- **`[num_features].mean()`**: Calculates the mean of each numerical feature for each group.
- **Purpose:**
  - Provides a quick statistical comparison between stroke and non-stroke individuals.
  - Helps identify features that may be strong predictors of stroke.

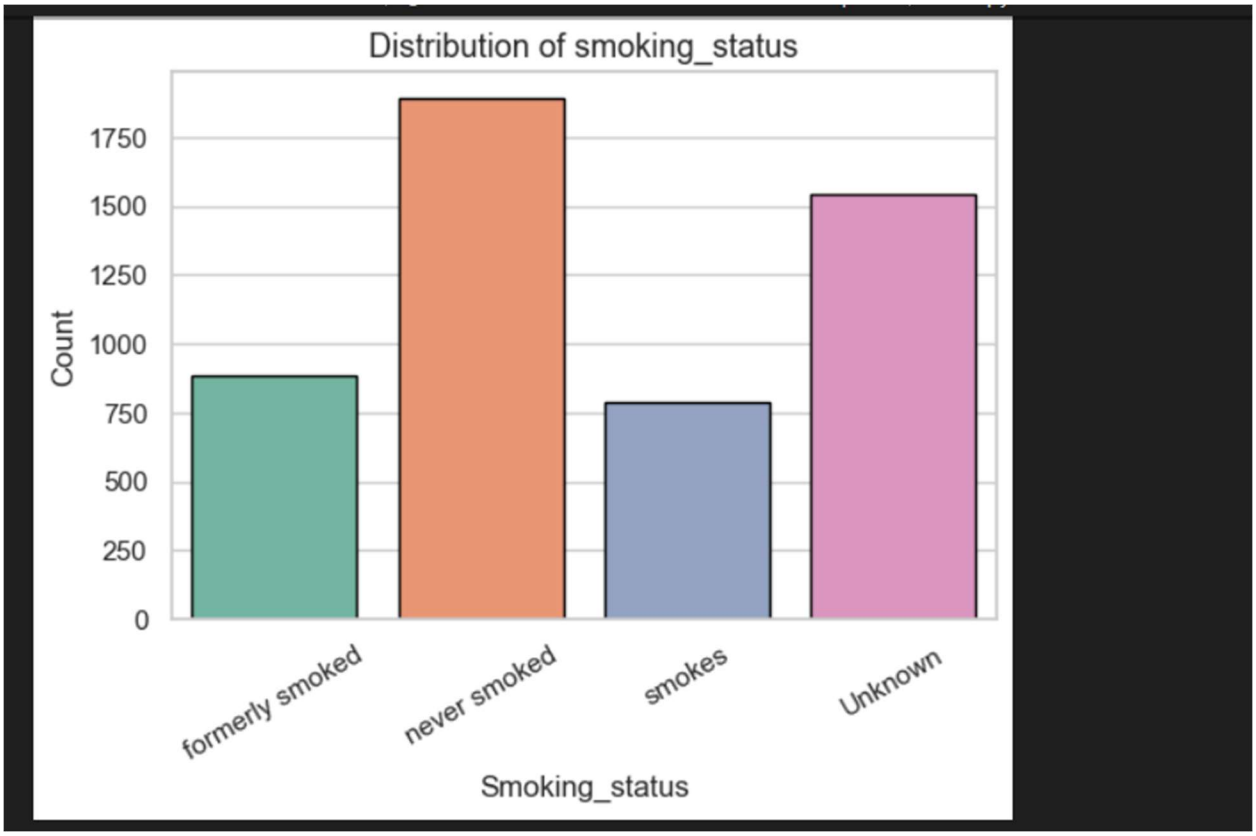
Example Interpretation:

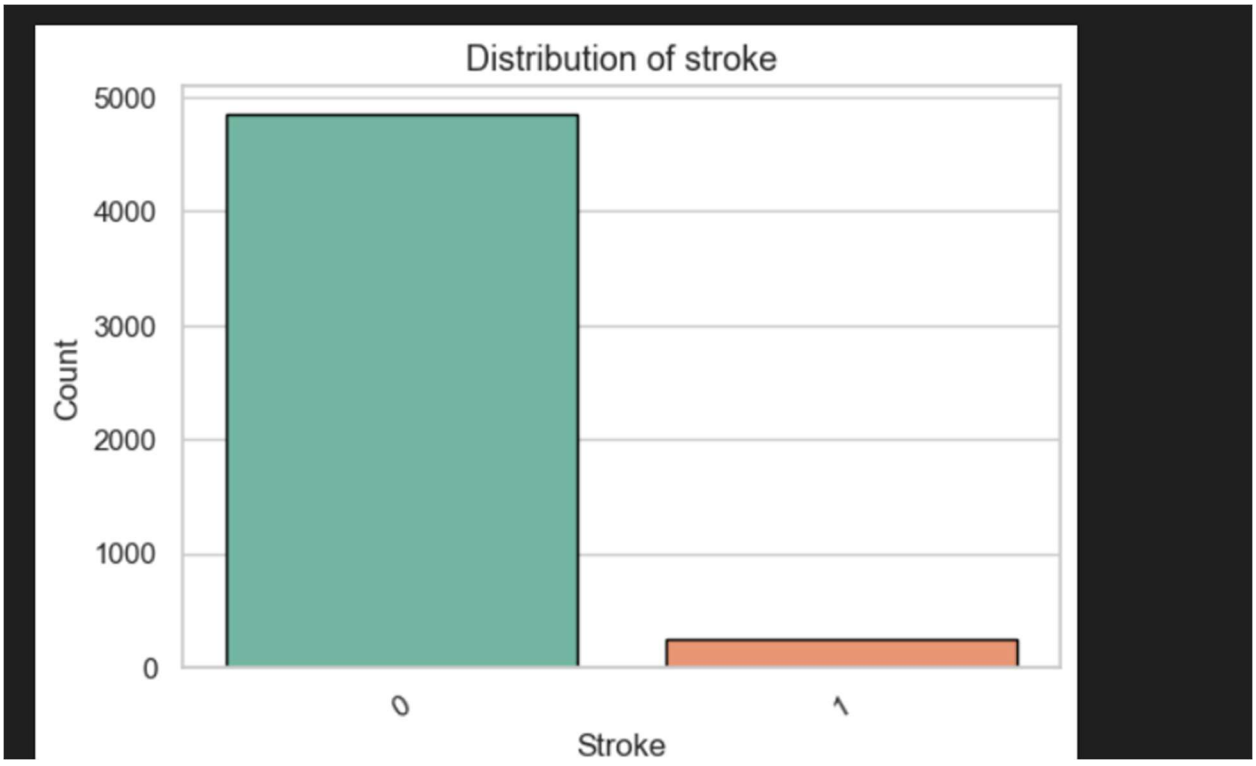
- If the mean age of stroke patients is higher than non-stroke patients, age may be a strong risk factor.
- Similarly, higher average glucose levels or BMI in stroke cases may indicate relevant health patterns.

```
cat_features_3 = ['Residence_type', 'smoking_status', 'stroke']

for col in cat_features_3:
    plt.figure(figsize=(6,4))
    sns.countplot(x=col, data=df, palette='Set2', edgecolor='black', hue=None, legend=False)
    plt.title(f'Distribution of {col}', fontsize=13)
    plt.xlabel(col.capitalize())
    plt.ylabel('Count')
    plt.xticks(rotation=30)
```

```
plt.show()
```

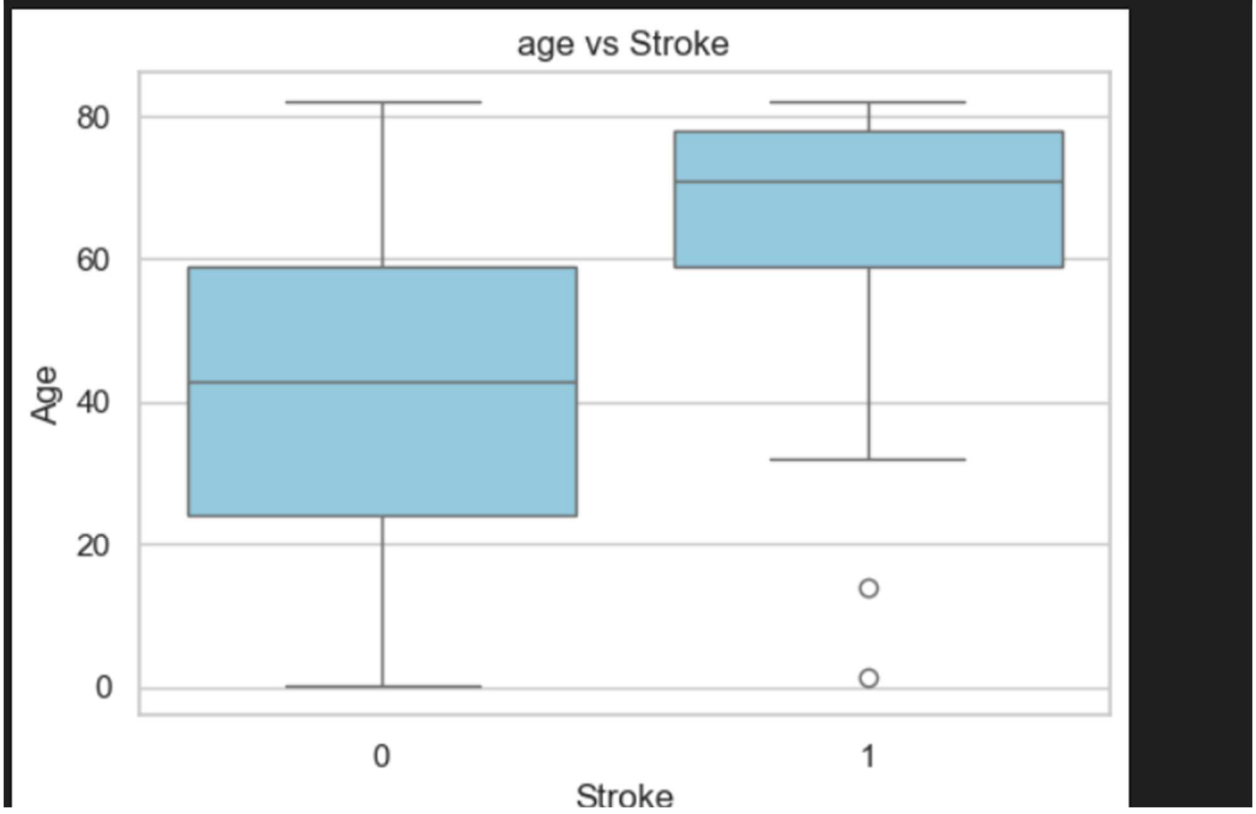

Distribution of Residence_type
Count
2500
2000
1500
1000
500
0
Urban
Rural
Residence_type

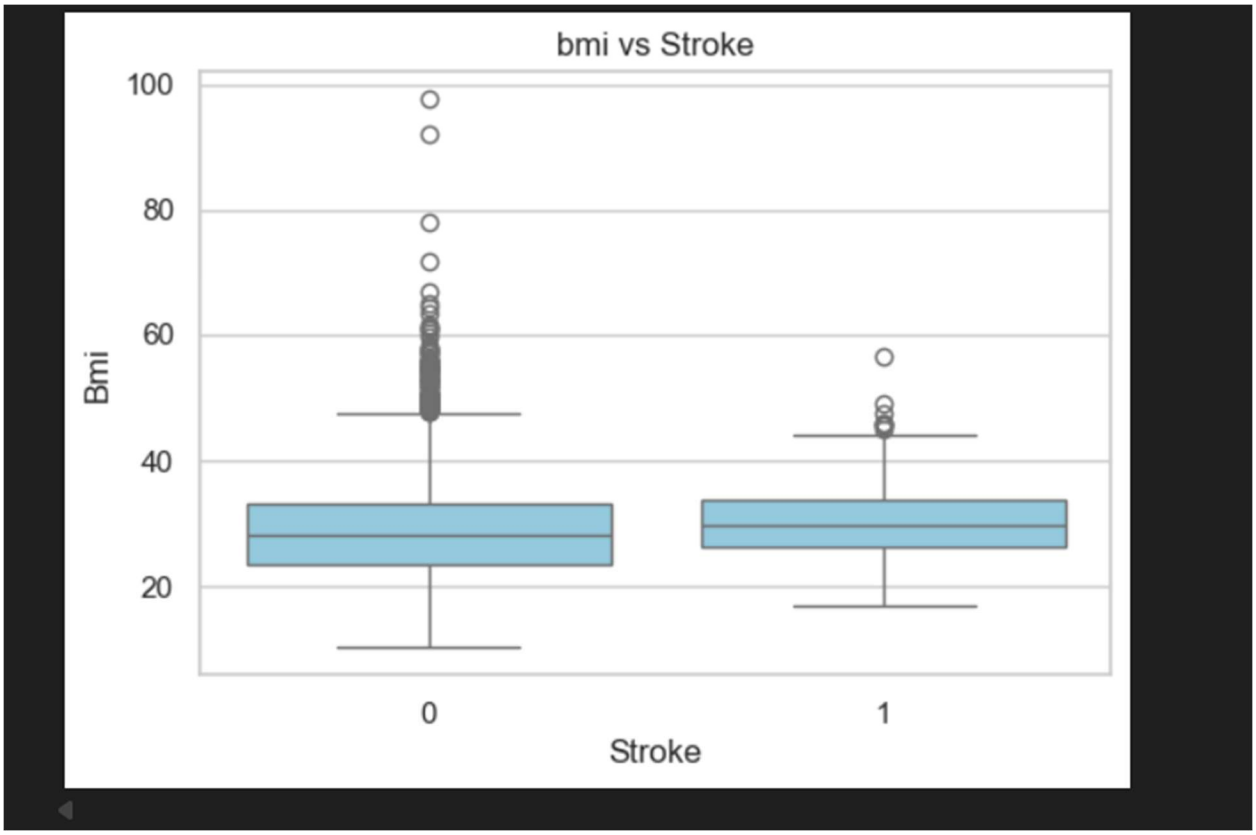


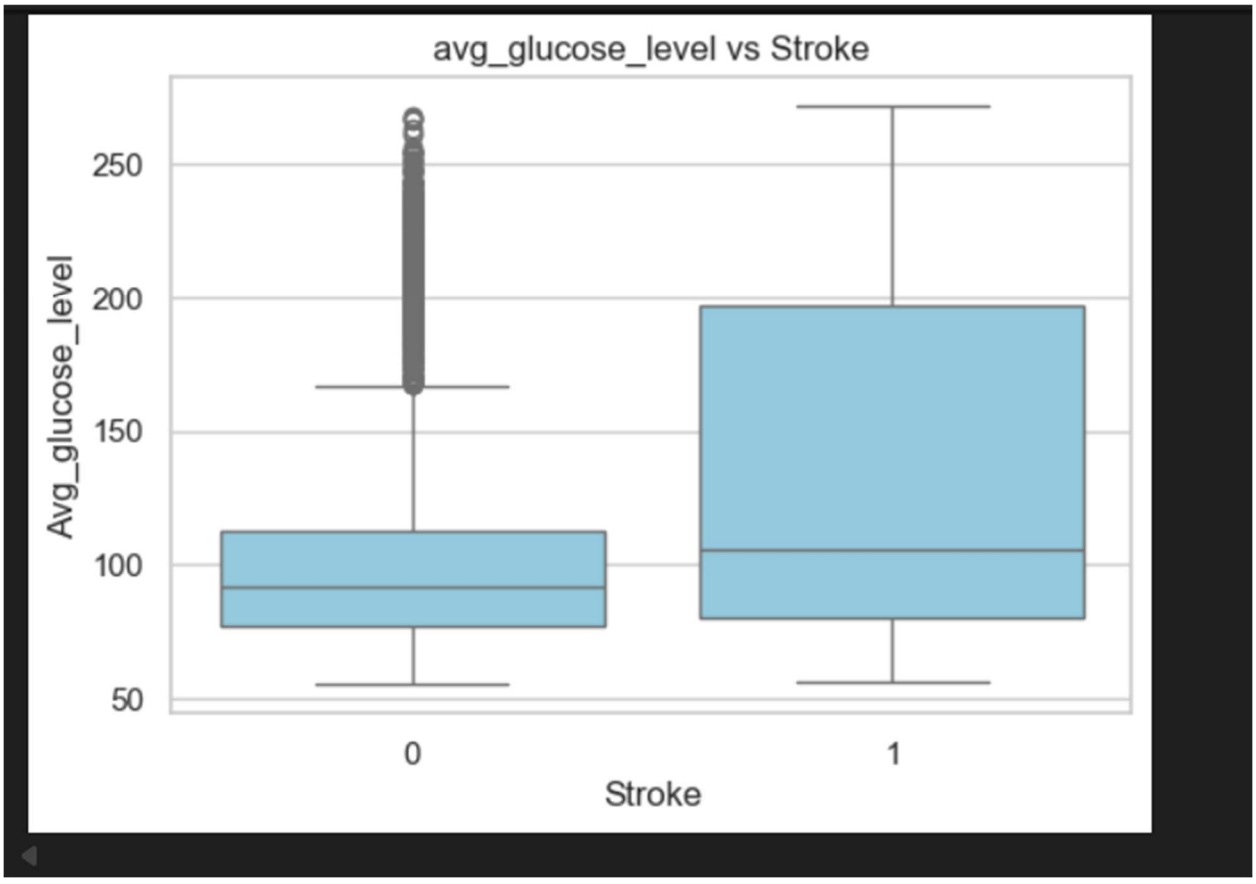



```
num_features = ['age', 'bmi', 'avg_glucose_level']
```

```
for col in num_features:
    plt.figure(figsize=(6,4))
    sns.boxplot(x='stroke', y=col, data=df, color='skyblue')  # Use color instead
of palette
    plt.title(f'{col} vs Stroke')
    plt.xlabel('Stroke')
    plt.ylabel(col.capitalize())
    plt.show()

# Mean comparison table
print(df.groupby('stroke')[num_features].mean())
```




bmi vs Stroke
Bmi
100
80
60
40
20
0
1
Stroke


avg_glucose_level vs Stroke
Avg_glucose_level
250
200
150
100
50
0
1
Stroke

stroke			
0	41.971545	28.823064	104.795513
1	67.728193	30.471292	132.544739

INSIGHTS

old people have significantly higher risk of stroke

bmi doesnt seem to have large effect on stroke

people who have have higher glucose level tends to also have a stroke but need more analysis

📌 Bivariate Analysis: Categorical Features vs Stroke

Bivariate analysis examines the relationship between **two variables**. In this section, we analyze how categorical features relate to the target variable (**stroke**) to identify potential risk factors and trends.

Code

```
cat_features = ['gender', 'hypertension', 'heart_disease', 'ever_married',
                'work_type', 'Residence_type', 'smoking_status']

for col in cat_features:
    cross_tab = pd.crosstab(df[col], df['stroke'], normalize='index') * 100
    cross_tab.plot(kind='bar', stacked=True, figsize=(6,4),
colormap='Paired')
    plt.title(f'Stroke by {col}')
    plt.ylabel('Percentage (%)')
    plt.xlabel(col.capitalize())
    plt.xticks(rotation=30)
    plt.legend(title='Stroke', labels=['No', 'Yes'])
    plt.show()
```

📌 Code Explanation

1. Selecting categorical features

The list `cat_features` includes all relevant categorical predictors such as `gender`, `hypertension`, `work_type`, and `smoking_status`.

2. `pd.crosstab()`

- Creates a **cross-tabulation table** between the categorical feature and the target variable (`stroke`).

- `normalize='index'` converts counts to **row-wise percentages**, so each category shows the proportion of stroke and non-stroke cases.
- Multiplying by 100 converts proportions to **percentages**.

3. `cross_tab.plot(kind='bar', stacked=True, ...)`

- Plots a **stacked bar chart** showing the percentage of stroke vs non-stroke for each category.
- `colormap='Paired'` adds color distinction for better visualization.
- `stacked=True` allows easier comparison within each category.

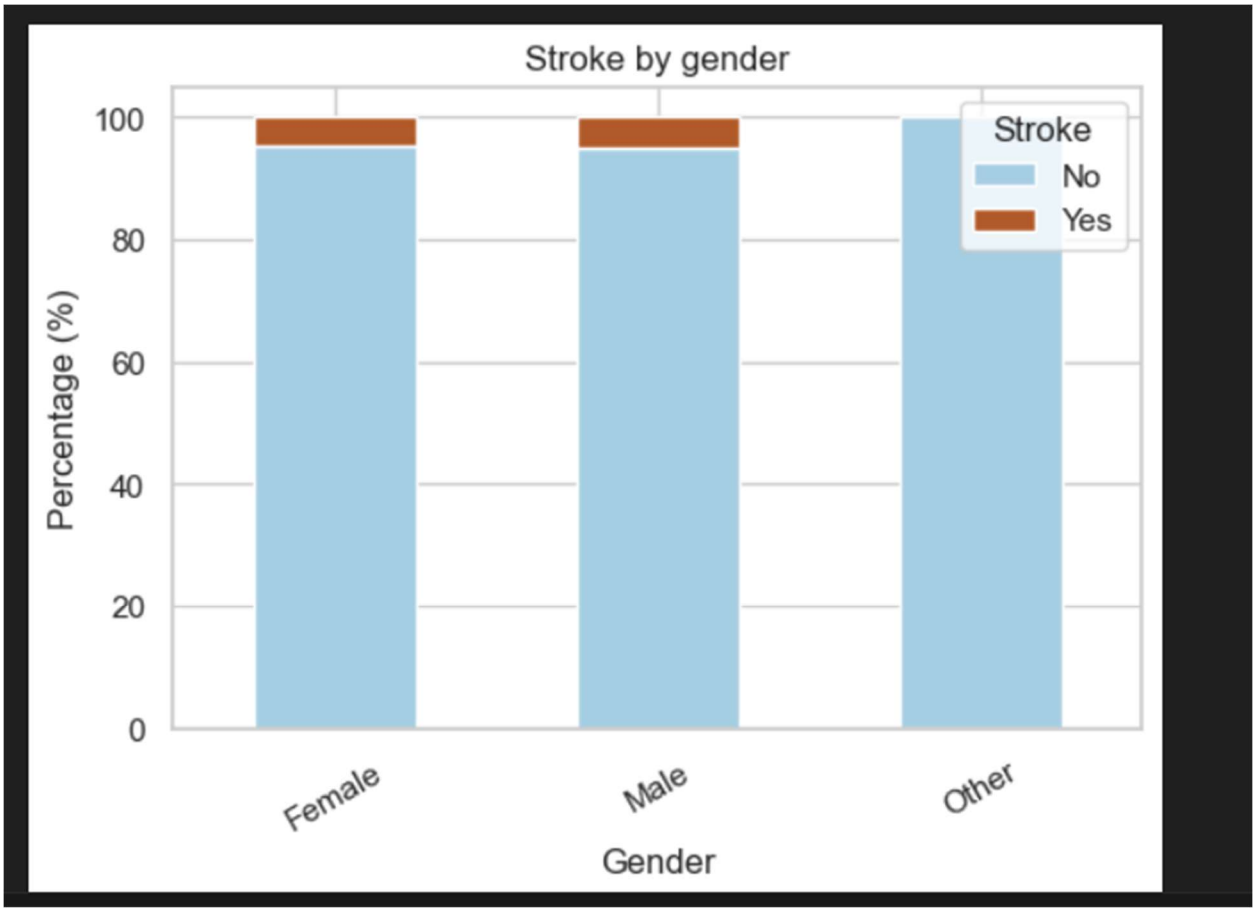
4. Labels and formatting

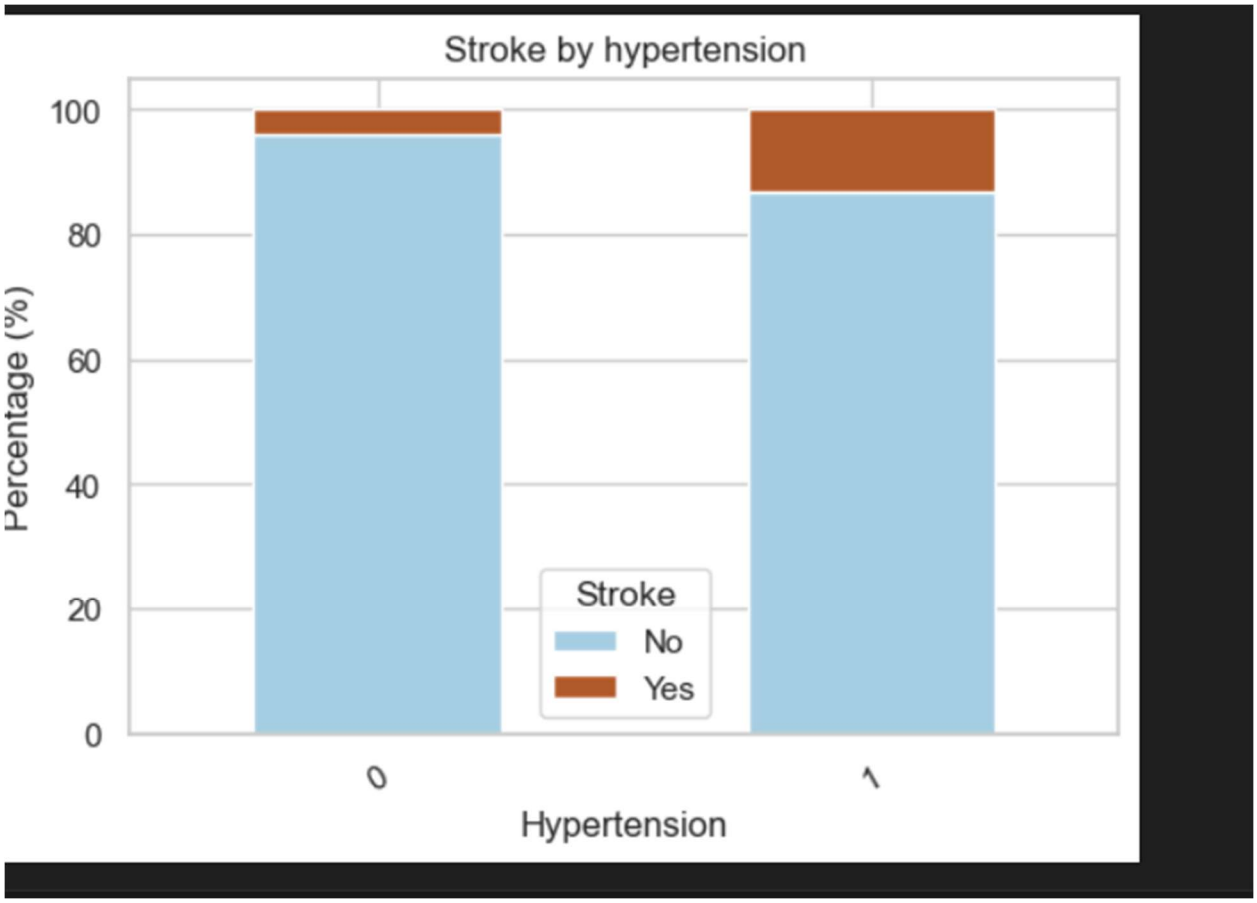
- Titles and axis labels make plots easy to interpret.
- Rotation of x-axis labels improves readability.
- Legend clearly shows which section represents stroke = 0 (No) and stroke = 1 (Yes).

📌 Purpose of This Analysis

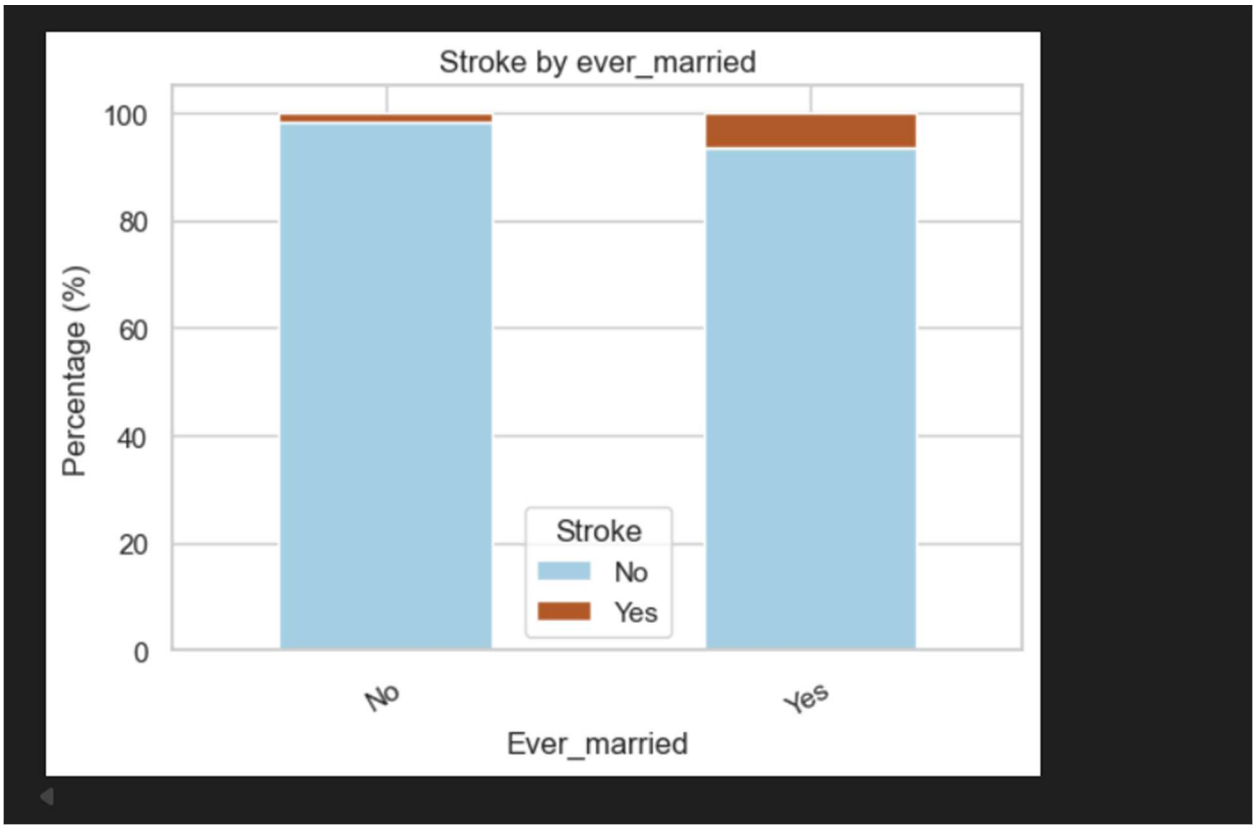
- To **visually assess** the impact of categorical features on the likelihood of stroke.
- To identify **high-risk categories** (e.g., people with hypertension or heart disease may have a higher percentage of stroke).
- Helps in **feature selection** and informs preprocessing decisions like encoding, grouping rare categories, or creating binary indicators.

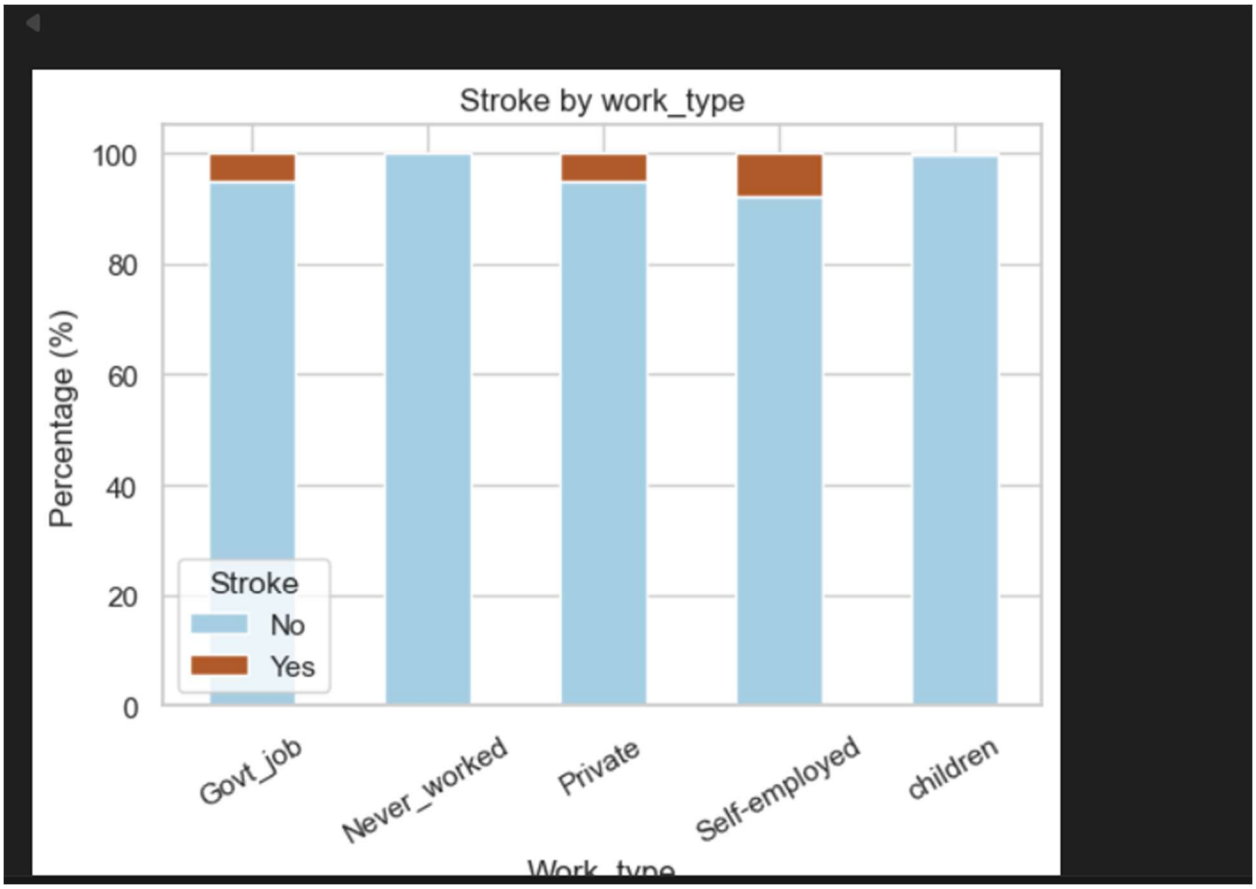
```
cat_features = ['gender', 'hypertension', 'heart_disease', 'ever_married',
                'work_type', 'Residence_type', 'smoking_status']

for col in cat_features:
    cross_tab = pd.crosstab(df[col], df['stroke'], normalize='index') *
100
    cross_tab.plot(kind='bar', stacked=True, figsize=(6,4),
colormap='Paired')
    plt.title(f'Stroke by {col}')
    plt.ylabel('Percentage (%)')
    plt.xlabel(col.capitalize())
    plt.xticks(rotation=30)
    plt.legend(title='Stroke', labels=['No', 'Yes'])
    plt.show()

```

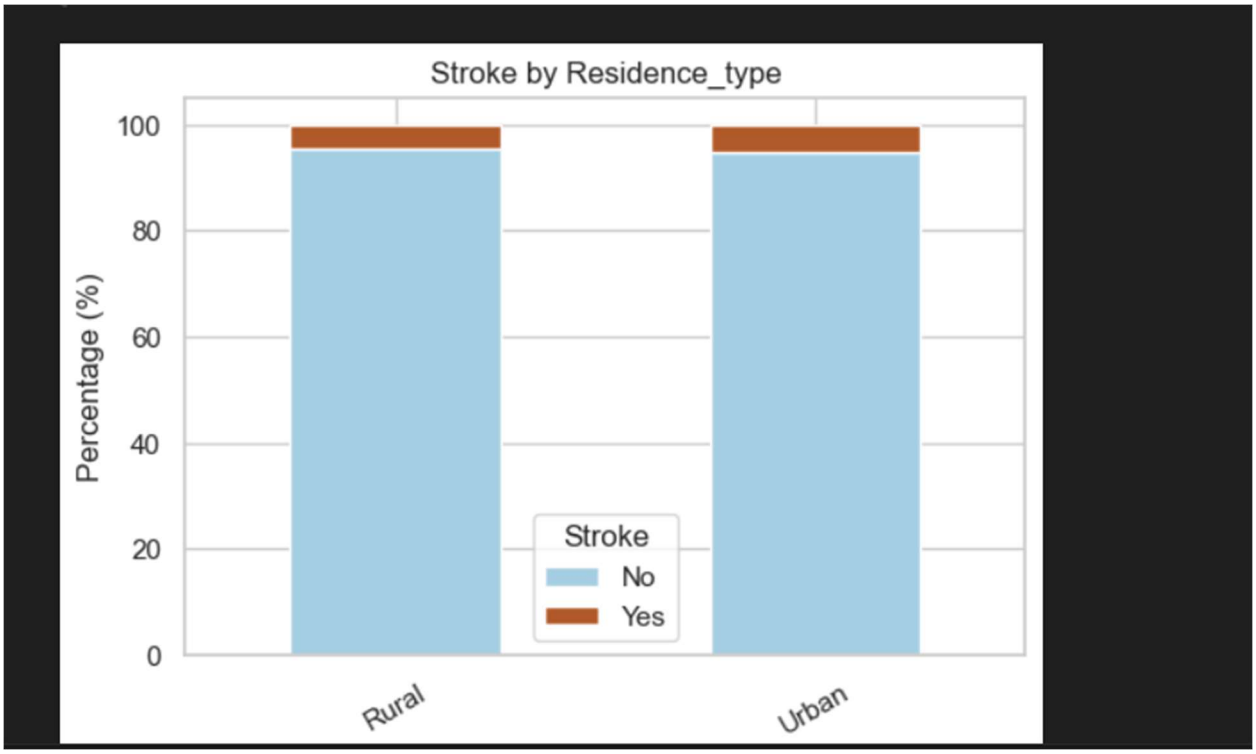

Stroke by gender
Stroke
No
Yes
Percentage (%)
100
80
60
40
20
0
Female
Male
Other
Gender

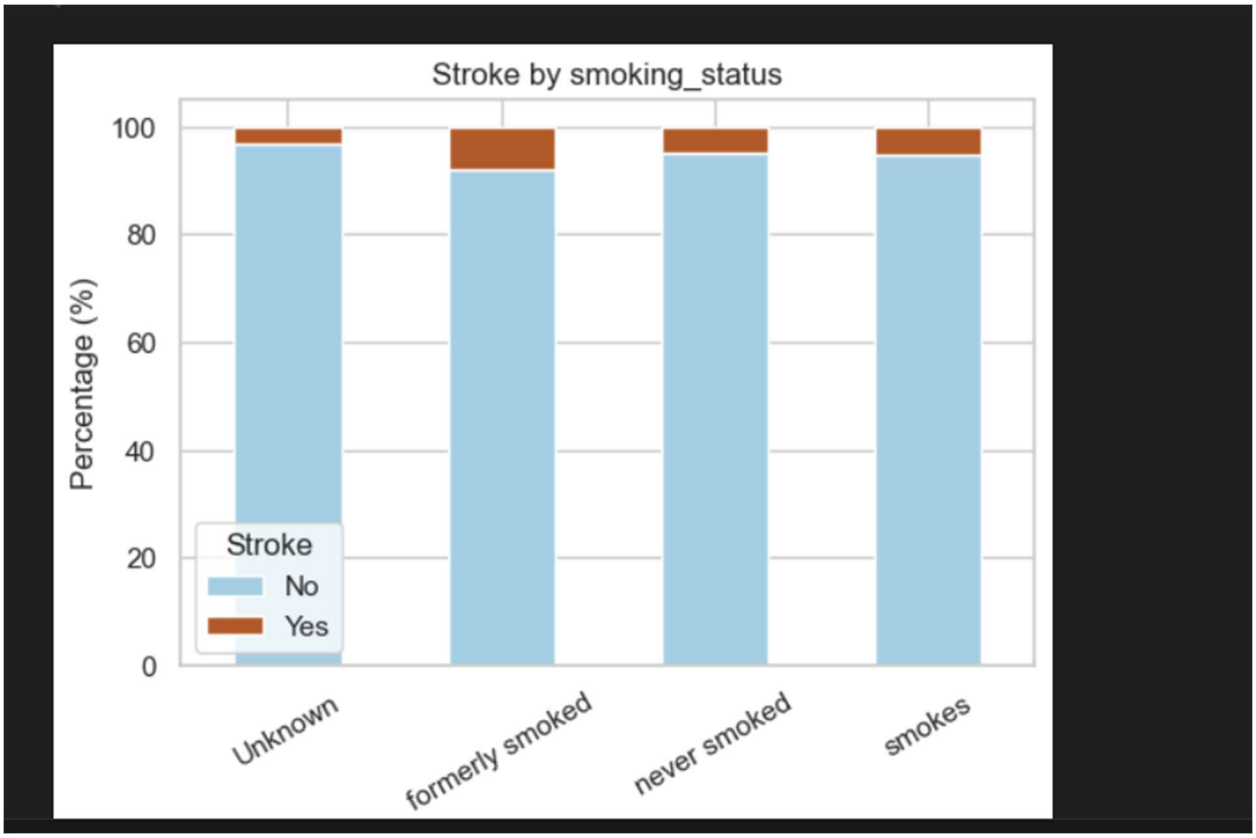

Stroke by hypertension
100
80
60
40
20
0
Percentage (%)
Stroke
No
Yes
0
1
Hypertension


Stroke by heart_disease
Percentage (%)
100
80
60
40
20
0
Stroke
No
Yes
0
1
Heart_disease


Stroke by ever_married
Percentage (%)
100
80
60
40
20
0
Stroke
No
Yes
No
Yes
Ever_married


Stroke by work_type
Percentage (%)
100
80
60
40
20
0
Stroke
No
Yes
Govt_job
Never_worked
Private
Self-employed
children
Work_type


Stroke by Residence_type
Percentage (%)
100
80
60
40
20
0
Stroke
No
Yes
Rural
Urban



INSIGHTS

Gender doesnt seem to have any direct effect with stroke

people with hpyertension have more stroke as it appears

people with heartdisesas also have more strokes

strangely marriage seems to have a relation with strokes as people who gets married have more stroke then single people this need more anlaysis

work type : people who work in government ,private or slef_employed have more chance of stroke than other work types

people living in urban have higher chance of haveing a stroke than people who live in rural but not significanlty hihger

frequent smoker have high chance of ahving a stroke

📌 Correlation Analysis

Correlation analysis is a **multivariate analysis technique** that measures the strength and direction of the relationship between numerical variables. Understanding correlations helps identify features that are strongly associated with the target variable (**stroke**) and can guide feature selection for predictive modeling.

Code

```
# Compute correlation matrix for all numeric features
corr = df.corr(numeric_only=True)
```

```
# Visualize correlation using a heatmap
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()

# Check correlation of each feature with the target
print(corr['stroke'].sort_values(ascending=False))

# Select numerical features including the target
num_features = ['age', 'bmi', 'avg_glucose_level', 'hypertension',
'heart_disease', 'stroke']

# Compute correlation matrix
corr_matrix = df[num_features].corr()

# Show correlation matrix
print(corr_matrix)

# Correlation of each numeric feature with the target
corr_with_target = corr_matrix['stroke'].sort_values(ascending=False)
print("Correlation with Stroke:\n", corr_with_target)
```

📌 Code Explanation

1. `df.corr(numeric_only=True)`

- Computes the **Pearson correlation coefficient** between all numerical features.
- Values range from **-1 to +1**:
  - **+1** → perfect positive correlation
  - **-1** → perfect negative correlation
  - **0** → no linear correlation

2. Heatmap Visualization

```
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
```

- Displays the correlation matrix visually.
- **`annot=True`** shows numeric values in each cell.
- **`cmap='coolwarm'`** colors positive correlations in warm tones and negative in cool tones.
- **`linewidths=0.5`** adds grid lines for clarity.

3. Correlation with the Target

```
print(corr['stroke'].sort_values(ascending=False))
```

- Extracts the correlation of each feature specifically with **stroke**.
- Sorting helps quickly identify features with strong positive or negative associations.

4. Focused Numerical Features

```
num_features = ['age', 'bmi', 'avg_glucose_level', 'hypertension',
'heart_disease', 'stroke']
```

- Includes both continuous and binary numerical features.
- Computing correlations only among these ensures a clear comparison with the target.

5. Correlation Matrix & Target Correlation

```
corr_matrix = df[num_features].corr()
corr_with_target = corr_matrix['stroke'].sort_values(ascending=False)
```

- Shows correlations among selected numerical features.
- Identifies which variables have the **strongest association with stroke**, aiding in feature selection.

📌 Purpose of Correlation Analysis

- Detect linear relationships between numerical variables.
- Identify **predictor variables** most associated with stroke.
- Reveal **multicollinearity**, which can affect regression-based models.
- Guide **feature selection and engineering** to improve model performance.

```
corr = df.corr(numeric_only=True)
plt.figure(figsize=(8,6))
sns.heatmap(corr, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()

# Check correlation with target
print(corr['stroke'].sort_values(ascending=False))
```


```
stroke               1.000000
age                  0.245257
heart_disease        0.134914
avg_glucose_level    0.131945
hypertension         0.127904
bmi                  0.042374
id                   0.006388
Name: stroke, dtype: float64
```

```
# Select numerical features including the target
num_features = ['age', 'bmi', 'avg_glucose_level', 'hypertension', 'heart_disease', 'stroke']

# Compute correlation matrix
corr_matrix = df[num_features].corr()

# Show correlation matrix
print(corr_matrix)
```

0.0s

```
# Select numerical features including the target
num_features = ['age', 'bmi', 'avg_glucose_level', 'hypertension',
'heart_disease', 'stroke']

# Compute correlation matrix
corr_matrix = df[num_features].corr()

# Show correlation matrix
```

```
print(corr_matrix)
```

```
…                       age       bmi  avg_glucose_level  hypertension  \
    age                1.000000  0.333398           0.238171      0.276398
    bmi                0.333398  1.000000           0.175502      0.167811
    avg_glucose_level  0.238171  0.175502           1.000000      0.174474
    hypertension       0.276398  0.167811           0.174474      1.000000
    heart_disease      0.263796  0.041357           0.161857      0.108306
    stroke             0.245257  0.042374           0.131945      0.127904

                       heart_disease    stroke
    age                     0.263796  0.245257
    bmi                     0.041357  0.042374
    avg_glucose_level       0.161857  0.131945
    hypertension            0.108306  0.127904
    heart_disease           1.000000  0.134914
    stroke                  0.134914  1.000000
```

```
# Correlation of each numeric feature with the target
corr_with_target = corr_matrix['stroke'].sort_values(ascending=False)
print("Correlation with Stroke:\n", corr_with_target)
```

```
…   Correlation with Stroke:
     stroke               1.000000
    age                   0.245257
    heart_disease         0.134914
    avg_glucose_level     0.131945
    hypertension          0.127904
    bmi                   0.042374
    Name: stroke, dtype: float64
```

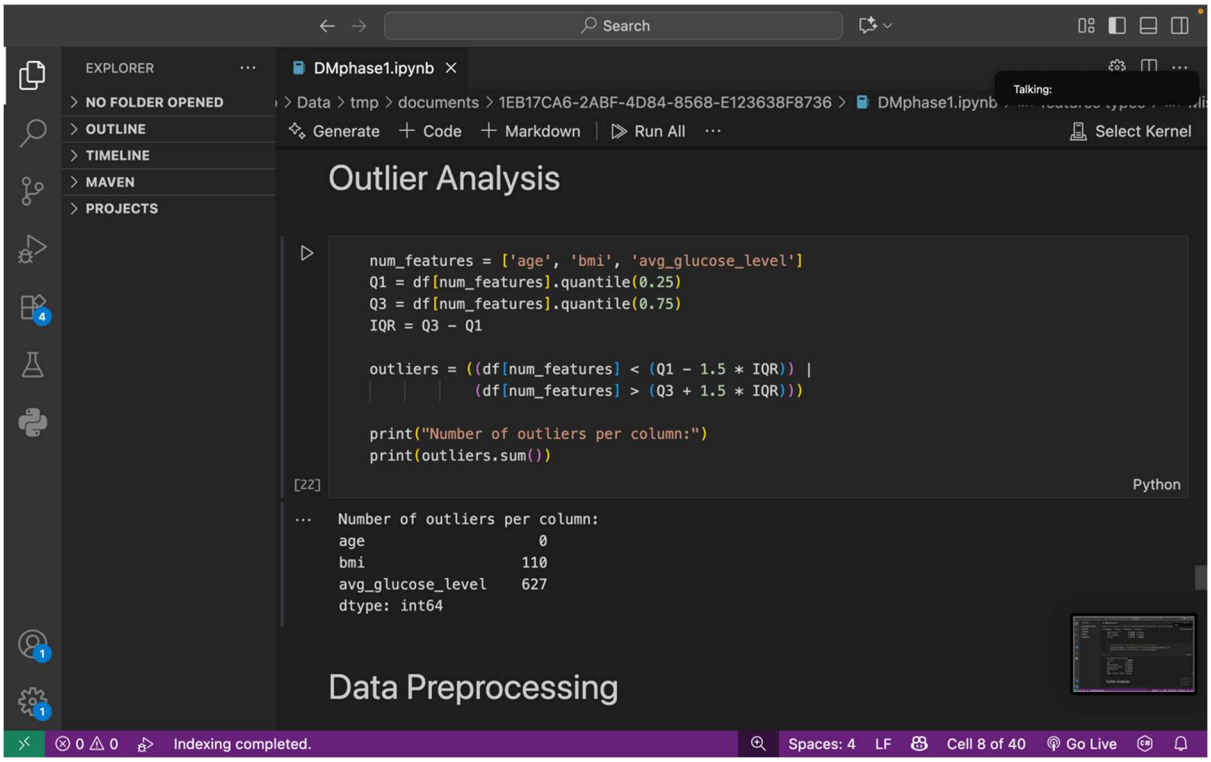
Outlier detection:

-takes selected numeric columns

-Computes Q1, Q3, and IQR

-Identifies outliers using the IQR rule

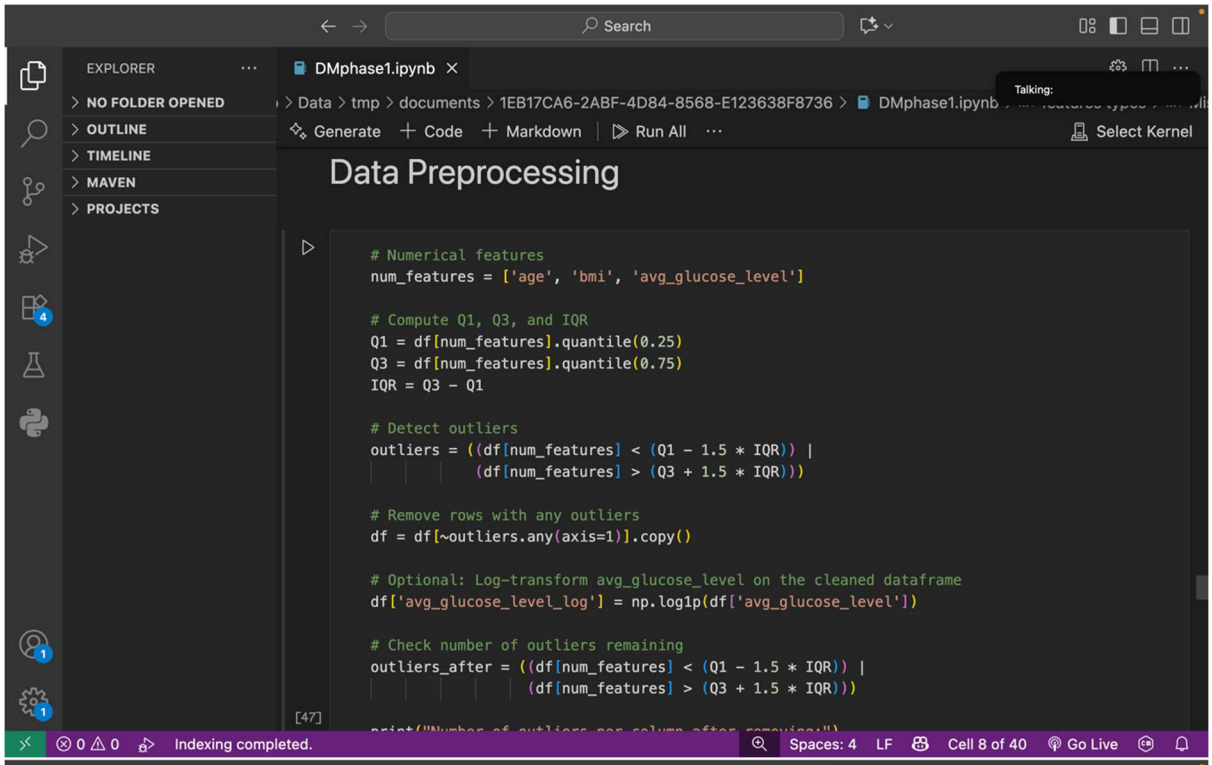
-Counts how many outliers appear in each column

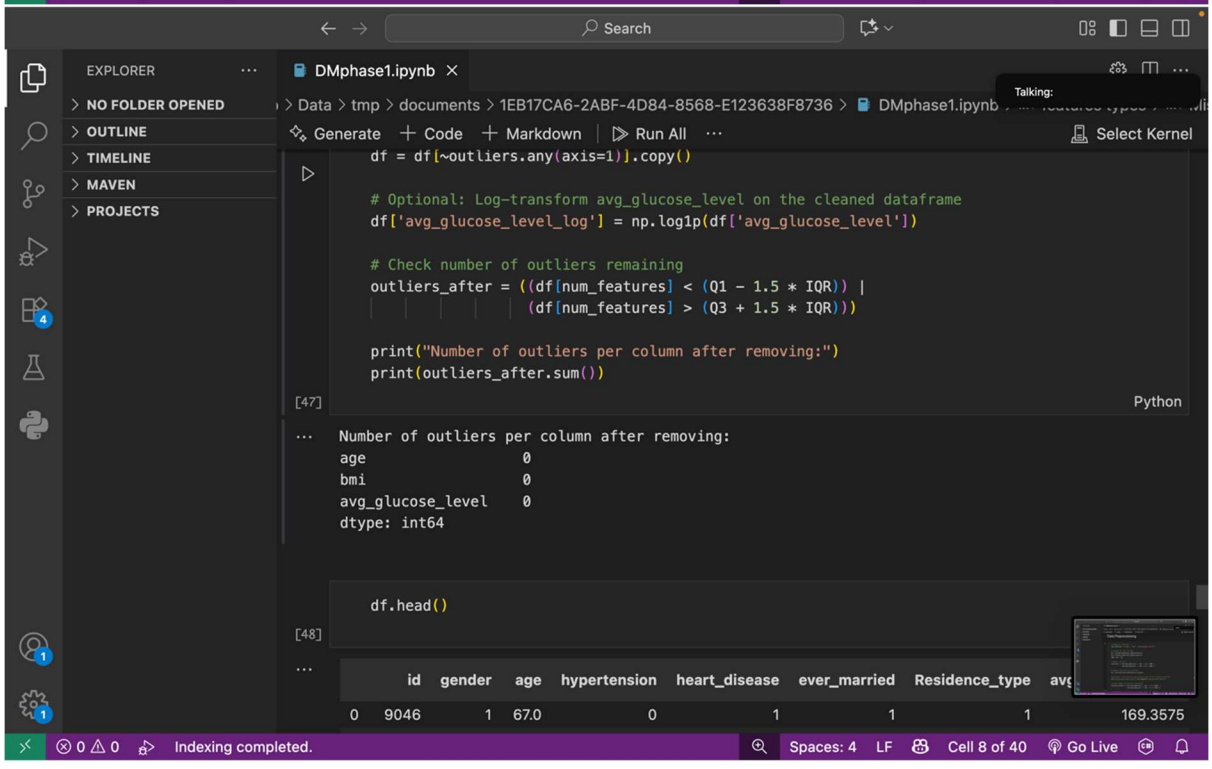


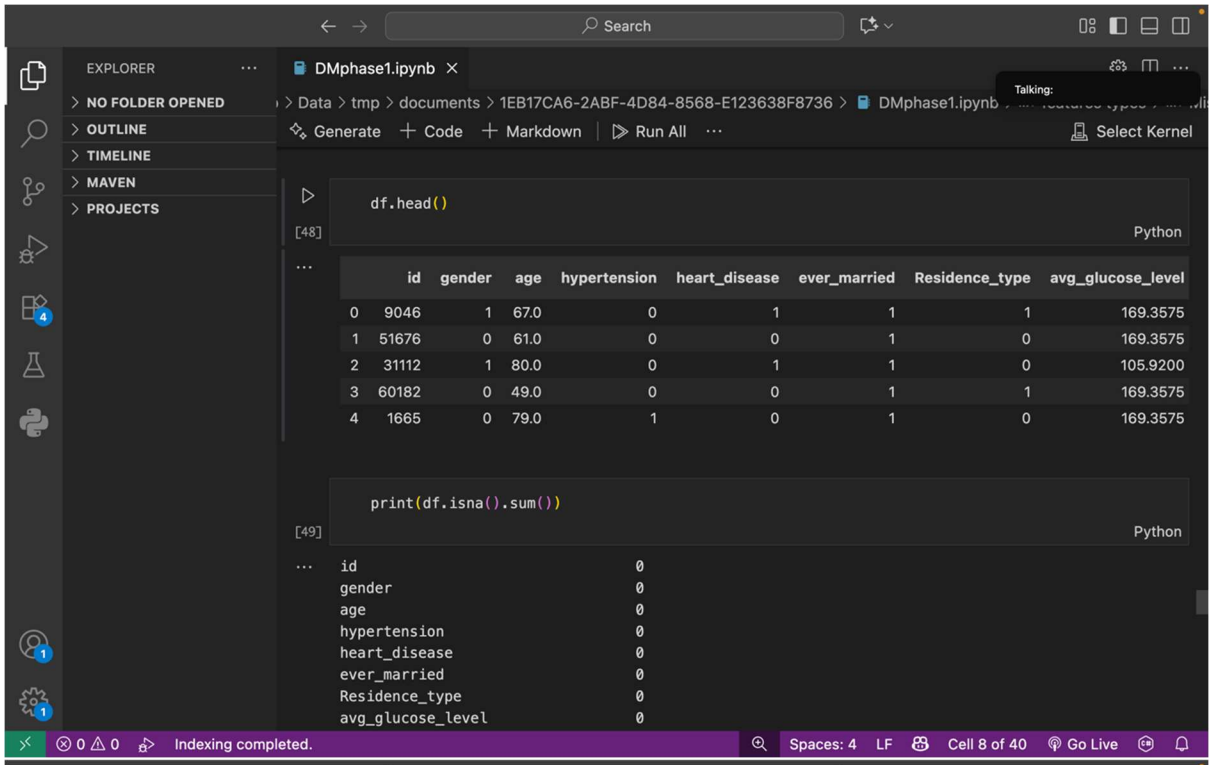
Data Processing:

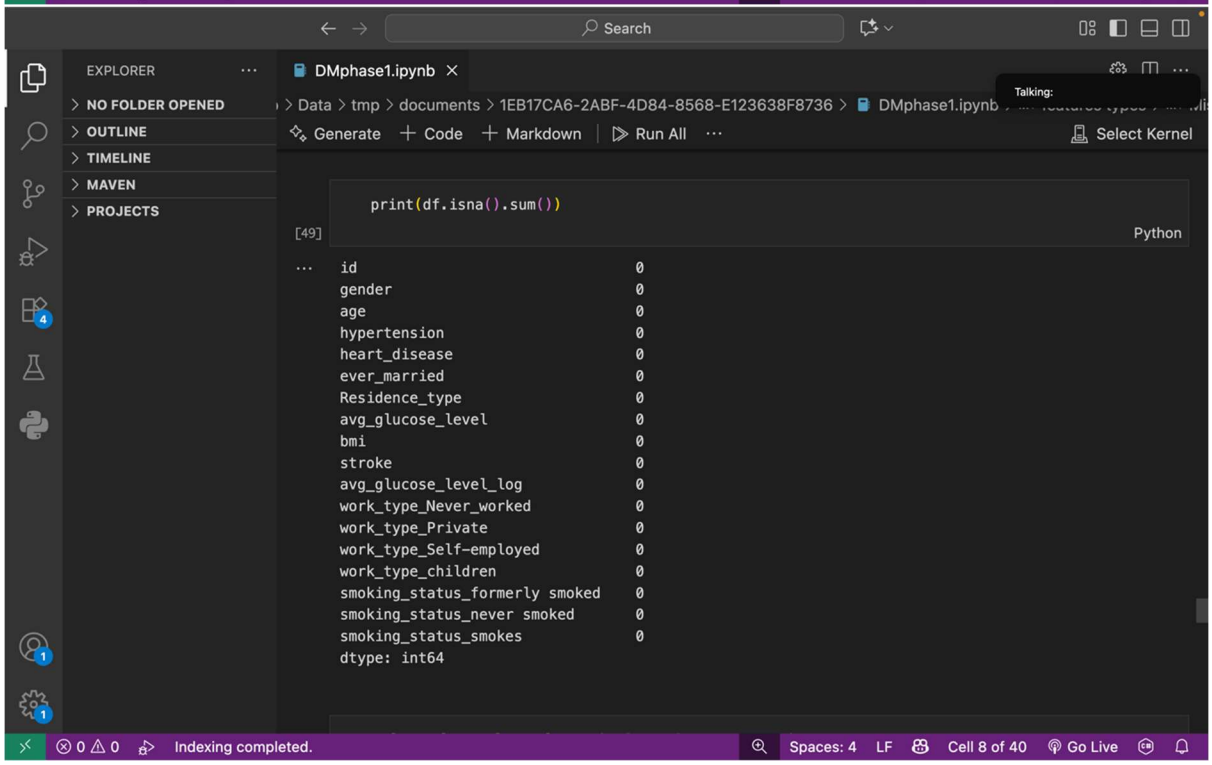
This code performs a full pre-processing pipeline for preparing a dataset for modelling. It begins by selecting the numerical columns (*age*, *bmi*, and *avg_glucose_level*) and identifying outliers using the IQR method by computing the 25th percentile (Q1), 75th percentile (Q3), and the interquartile range (IQR). Any rows containing values outside the range **Q1 – 1.5×IQR to Q3 + 1.5×IQR** are removed. After cleaning, a log transformation is applied to *avg_glucose_level* to reduce skewness. The code then fixes invalid ages by replacing any age below 1 with NaN and later filling missing age values with the median. Next, it handles categorical variables by filling missing values—using the mode when available or `"Unknown"` if the entire column is missing—then encoding binary categorical variables with Label Encoder and one-hot encoding multi-class categorical columns like *work type* and *smoking status*. Missing numeric values such as BMI are

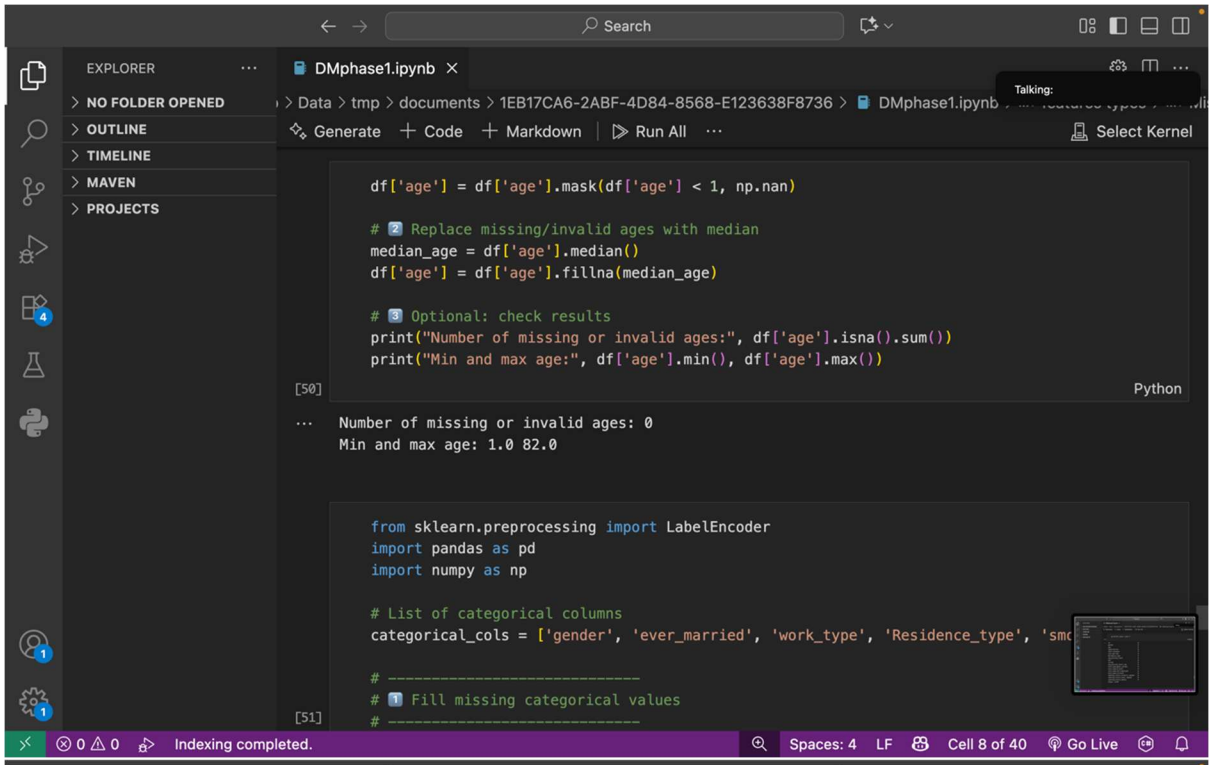
also filled with their median to maintain consistency. After pre-processing, the code prints missing values, data types, and a preview of the cleaned dataset. Finally, it visualizes the distribution of numerical features using histograms and calculates stroke rates for different work types and smoking statuses by computing the proportion of stroke cases within each encoded category, displaying these results in bar plots.

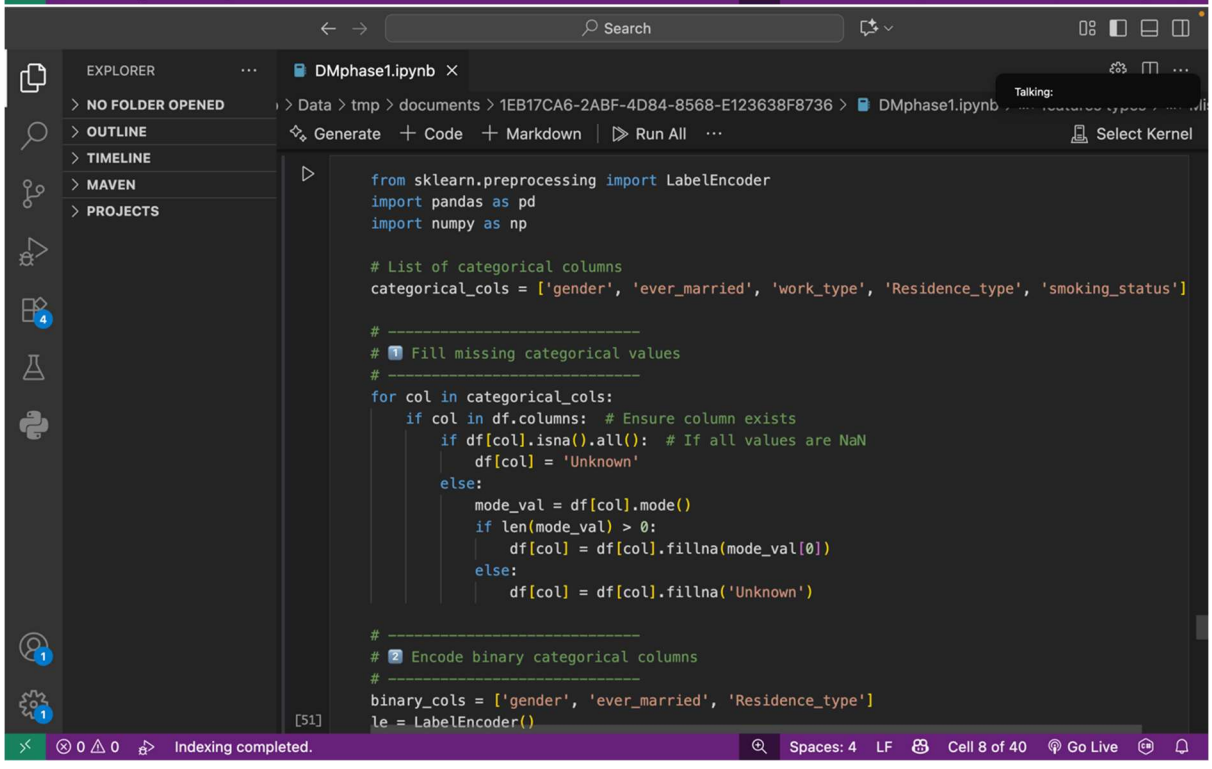

Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
Data Preprocessing
# Numerical features
num_features = ['age', 'bmi', 'avg_glucose_level']
# Compute Q1, Q3, and IQR
Q1 = df[num_features].quantile(0.25)
Q3 = df[num_features].quantile(0.75)
IQR = Q3 - Q1
# Detect outliers
outliers = ((df[num_features] < (Q1 - 1.5 * IQR)) |
(df[num_features] > (Q3 + 1.5 * IQR)))
# Remove rows with any outliers
df = df[~outliers.any(axis=1)].copy()
# Optional: Log-transform avg_glucose_level on the cleaned dataframe
df['avg_glucose_level_log'] = np.log1p(df['avg_glucose_level'])
# Check number of outliers remaining
outliers_after = ((df[num_features] < (Q1 - 1.5 * IQR)) |
(df[num_features] > (Q3 + 1.5 * IQR)))
[47]
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live

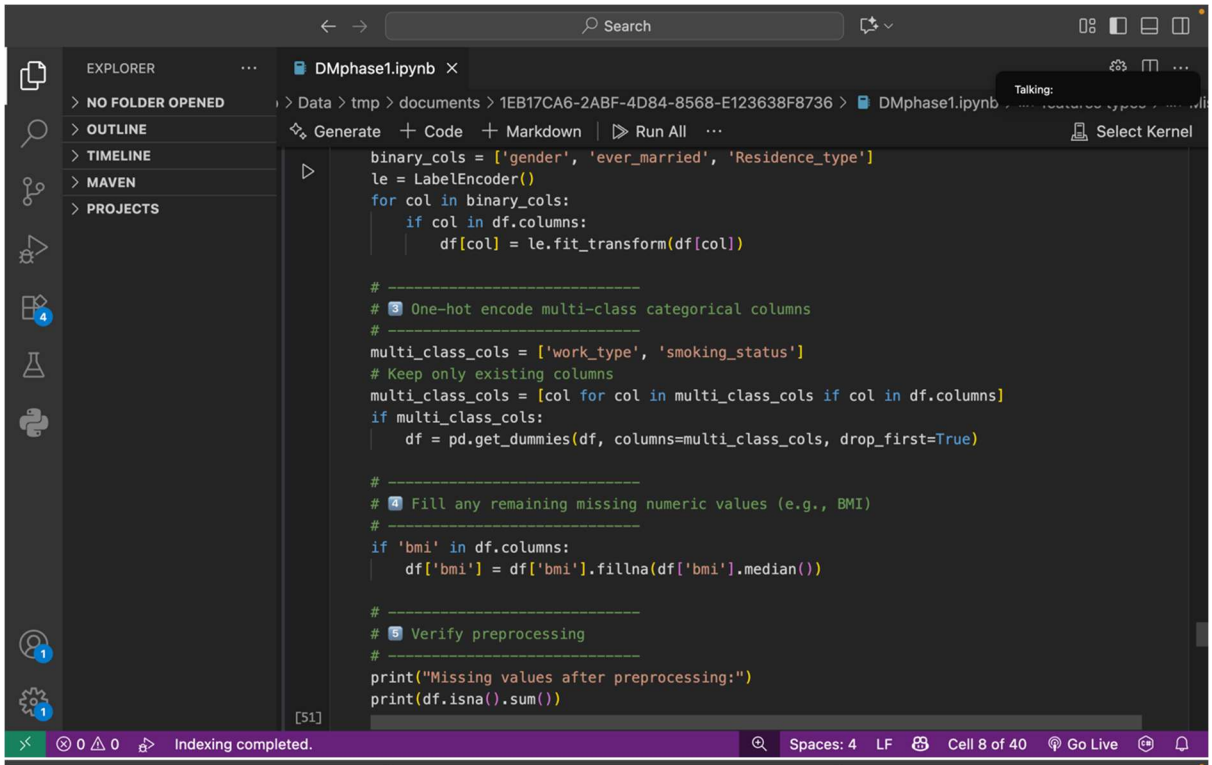

Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
df = df[~outliers.any(axis=1)].copy()
# Optional: Log-transform avg_glucose_level on the cleaned dataframe
df['avg_glucose_level_log'] = np.log1p(df['avg_glucose_level'])
# Check number of outliers remaining
outliers_after = ((df[num_features] < (Q1 - 1.5 * IQR)) |
(df[num_features] > (Q3 + 1.5 * IQR)))
print("Number of outliers per column after removing:")
print(outliers_after.sum())
[47]
Python
Number of outliers per column after removing:
age 0
bmi 0
avg_glucose_level 0
dtype: int64
df.head()
[48]
id gender age hypertension heart_disease ever_married Residence_type
0 9046 1 67.0 0 1 1 1 169.3575
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live

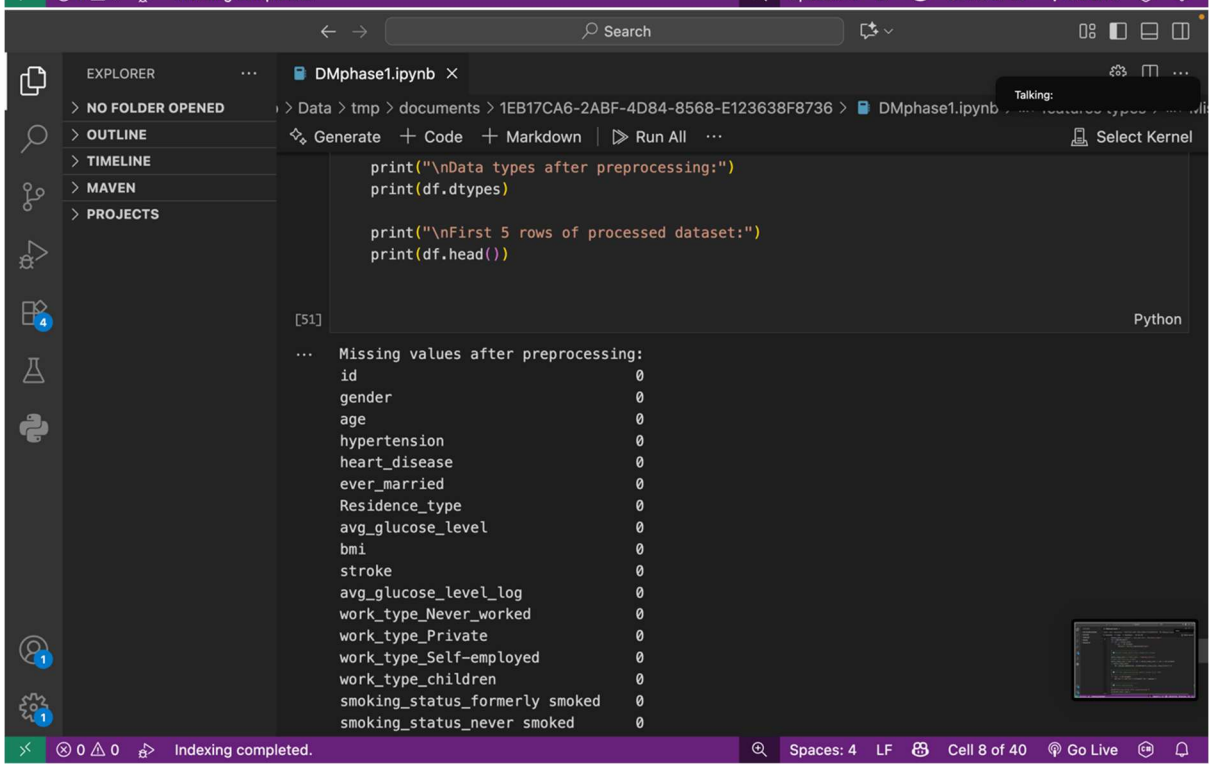

Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
[48]
df.head()
Python
id gender age hypertension heart_disease ever_married Residence_type avg_glucose_level
0 9046 1 67.0 0 1 1 1 169.3575
1 51676 0 61.0 0 0 1 0 169.3575
2 31112 1 80.0 0 1 1 0 105.9200
3 60182 0 49.0 0 0 1 1 169.3575
4 1665 0 79.0 1 0 1 0 169.3575
[49]
print(df.isna().sum())
Python
id 0
gender 0
age 0
hypertension 0
heart_disease 0
ever_married 0
Residence_type 0
avg_glucose_level 0
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live

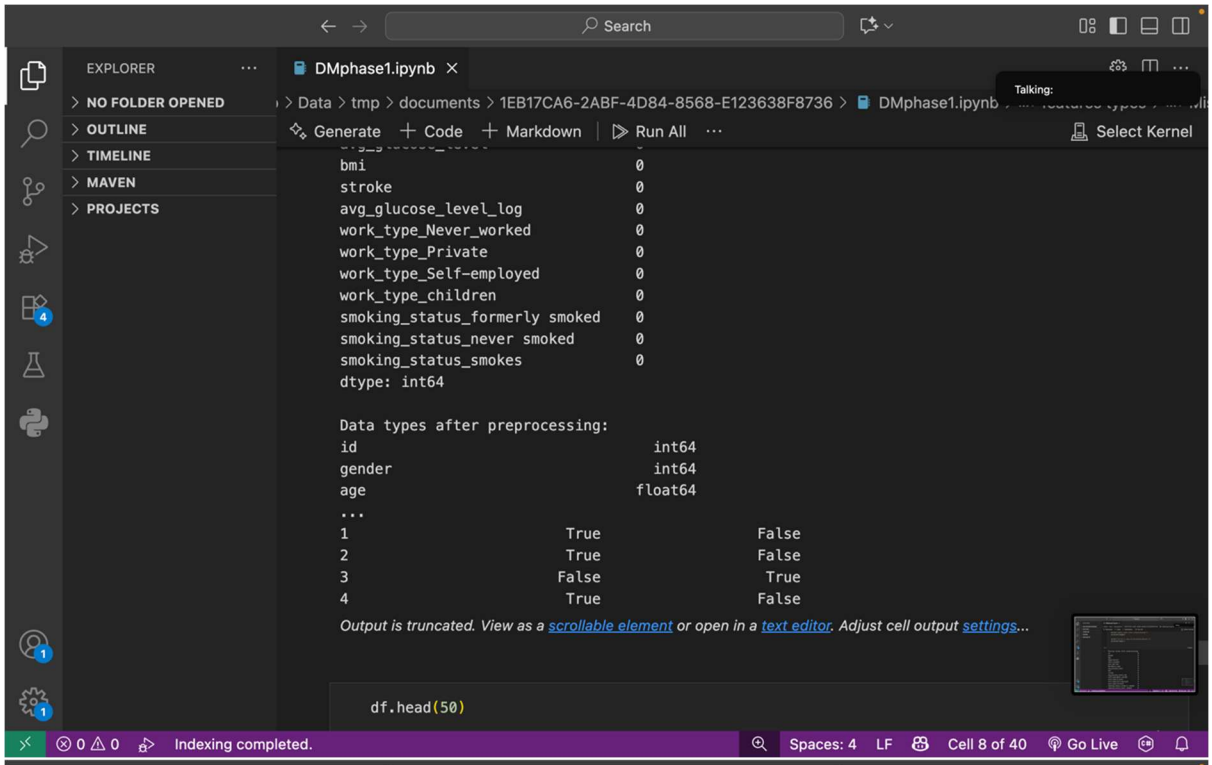

Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
[49]
print(df.isna().sum())
Python
id 0
gender 0
age 0
hypertension 0
heart_disease 0
ever_married 0
Residence_type 0
avg_glucose_level 0
bmi 0
stroke 0
avg_glucose_level_log 0
work_type_Never_worked 0
work_type_Private 0
work_type_Self-employed 0
work_type_children 0
smoking_status_formerly smoked 0
smoking_status_never smoked 0
smoking_status_smokes 0
dtype: int64
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live

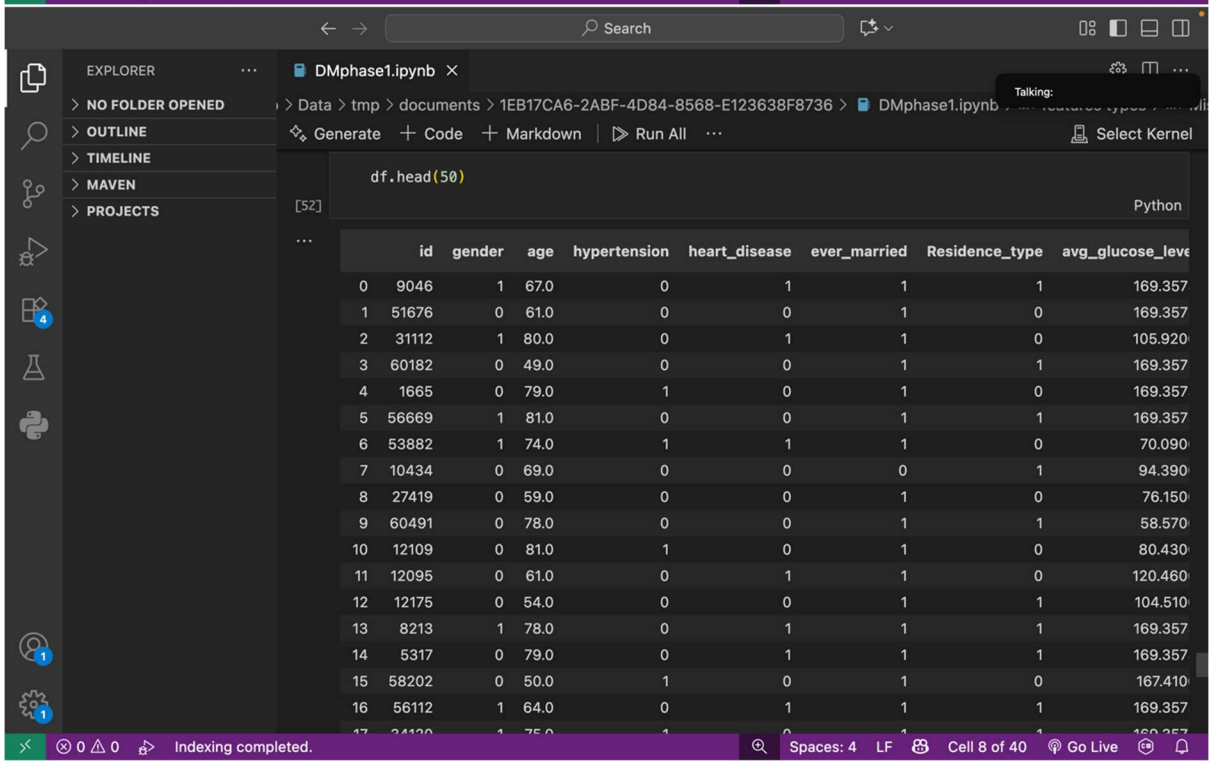

Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
df['age'] = df['age'].mask(df['age'] < 1, np.nan)
# 2 Replace missing/invalid ages with median
median_age = df['age'].median()
df['age'] = df['age'].fillna(median_age)
# 3 Optional: check results
print("Number of missing or invalid ages:", df['age'].isna().sum())
print("Min and max age:", df['age'].min(), df['age'].max())
[50]
Python
Number of missing or invalid ages: 0
Min and max age: 1.0 82.0
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
# List of categorical columns
categorical_cols = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smo
# ------------------------------
# 1 Fill missing categorical values
# ------------------------------
[51]
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live


Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
from sklearn.preprocessing import LabelEncoder
import pandas as pd
import numpy as np
# List of categorical columns
categorical_cols = ['gender', 'ever_married', 'work_type', 'Residence_type', 'smoking_status']
# ------------------------------
# 1 Fill missing categorical values
# ------------------------------
for col in categorical_cols:
if col in df.columns: # Ensure column exists
if df[col].isna().all(): # If all values are NaN
df[col] = 'Unknown'
else:
mode_val = df[col].mode()
if len(mode_val) > 0:
df[col] = df[col].fillna(mode_val[0])
else:
df[col] = df[col].fillna('Unknown')
# ------------------------------
# 2 Encode binary categorical columns
# ------------------------------
binary_cols = ['gender', 'ever_married', 'Residence_type']
le = LabelEncoder()
[51]
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live


Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
binary_cols = ['gender', 'ever_married', 'Residence_type']
le = LabelEncoder()
for col in binary_cols:
if col in df.columns:
df[col] = le.fit_transform(df[col])
# ----------------------------
# 3 One-hot encode multi-class categorical columns
# ----------------------------
multi_class_cols = ['work_type', 'smoking_status']
# Keep only existing columns
multi_class_cols = [col for col in multi_class_cols if col in df.columns]
if multi_class_cols:
df = pd.get_dummies(df, columns=multi_class_cols, drop_first=True)
# ----------------------------
# 4 Fill any remaining missing numeric values (e.g., BMI)
# ----------------------------
if 'bmi' in df.columns:
df['bmi'] = df['bmi'].fillna(df['bmi'].median())
# ----------------------------
# 5 Verify preprocessing
# ----------------------------
print("Missing values after preprocessing:")
print(df.isna().sum())
[51]
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live


Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Talking:
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Generate
Code
Markdown
Run All
Select Kernel
print("\nData types after preprocessing:")
print(df.dtypes)
print("\nFirst 5 rows of processed dataset:")
print(df.head())
[51]
Python
Missing values after preprocessing:
id 0
gender 0
age 0
hypertension 0
heart_disease 0
ever_married 0
Residence_type 0
avg_glucose_level 0
bmi 0
stroke 0
avg_glucose_level_log 0
work_type_Never_worked 0
work_type_Private 0
work_type_Self-employed 0
work_type_children 0
smoking_status_formerly smoked 0
smoking_status_never smoked 0
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live





	id	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_leve
0	9046	1	67.0	0	1	1	1	169.357
1	51676	0	61.0	0	0	1	0	169.357
2	31112	1	80.0	0	1	1	0	105.920
3	60182	0	49.0	0	0	1	1	169.357
4	1665	0	79.0	1	0	1	0	169.357
5	56669	1	81.0	0	0	1	1	169.357
6	53882	1	74.0	1	1	1	0	70.090
7	10434	0	69.0	0	0	0	1	94.390
8	27419	0	59.0	0	0	1	0	76.150
9	60491	0	78.0	0	0	1	1	58.570
10	12109	0	81.0	1	0	1	0	80.430
11	12095	0	61.0	0	1	1	0	120.460
12	12175	0	54.0	0	0	1	1	104.510
13	8213	1	78.0	0	1	1	1	169.357
14	5317	0	79.0	0	1	1	1	169.357
15	58202	0	50.0	1	0	1	0	167.410
16	56112	1	64.0	0	1	1	1	169.357

Search

EXPLORER

NO FOLDER OPENED

OUTLINE

TIMELINE

MAVEN

PROJECTS

DMphase1.ipynb

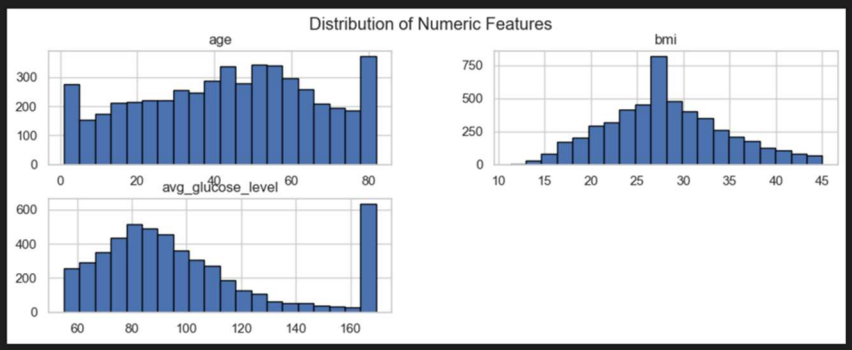
Talking:

Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb

Generate + Code + Markdown | Run All ... Select Kernel

```
numeric_cols = ['age', 'bmi', 'avg_glucose_level']

df[numeric_cols].hist(figsize=(12, 4), bins=20, edgecolor='black')
plt.suptitle("Distribution of Numeric Features")
plt.show()
```

[53] Python



0 0 Indexing completed. Spaces: 4 LF Cell 8 of 40 Go Live

Search

EXPLORER

NO FOLDER OPENED

OUTLINE

TIMELINE

MAVEN

PROJECTS

DMphase1.ipynb

Talking:

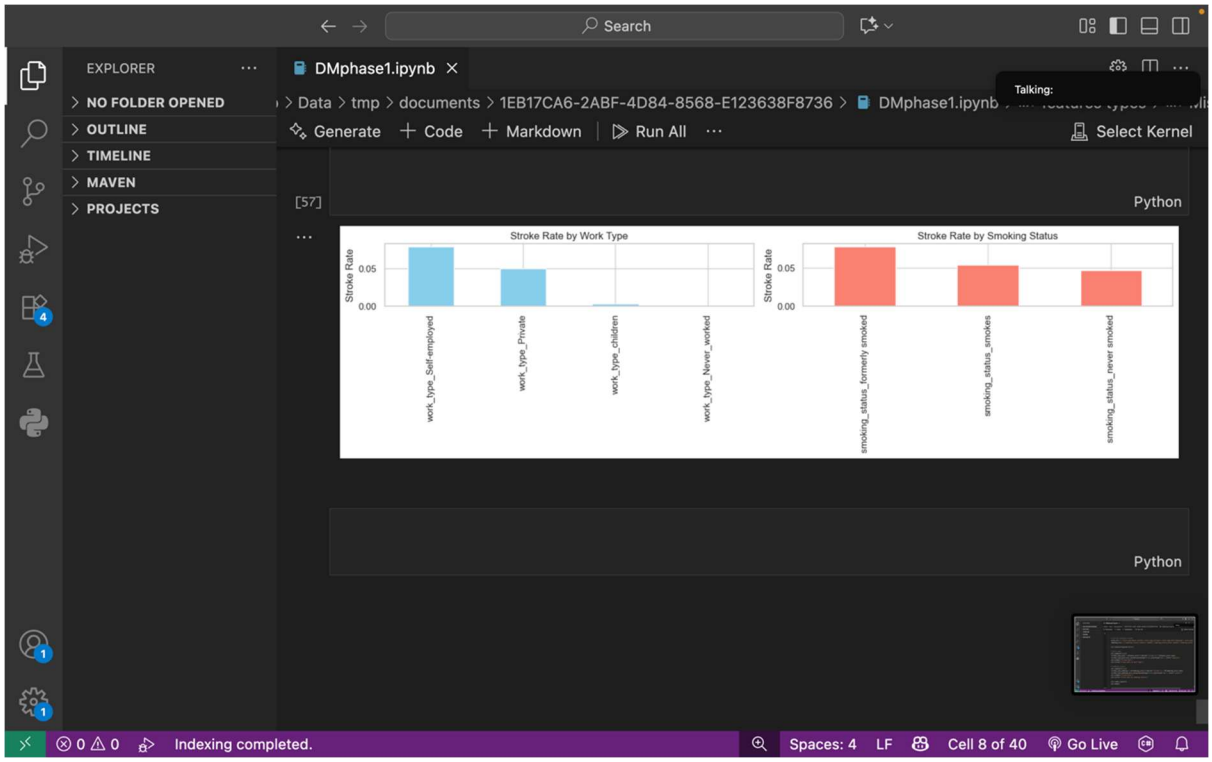
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb

Generate + Code + Markdown | Run All ... Select Kernel

```
# One-hot encoded columns
work_cols = ['work_type_Never_worked','work_type_Private','work_type_Self-employed','work_type
smoking_cols = ['smoking_status_formerly smoked','smoking_status_never smoked','smoking_status

plt.figure(figsize=(14,4))

# Work type
plt.subplot(1,2,1)
stroke_rate_work = df[work_cols].T.dot(df['stroke']) / df[work_cols].sum()
stroke_rate_work.sort_values(ascending=False).plot(kind='bar', color='skyblue')
plt.ylabel('Stroke Rate')
plt.title('Stroke Rate by Work Type')

# Smoking status
plt.subplot(1,2,2)
stroke_rate_smoking = df[smoking_cols].T.dot(df['stroke']) / df[smoking_cols].sum()
stroke_rate_smoking.sort_values(ascending=False).plot(kind='bar', color='salmon')
plt.ylabel('Stroke Rate')
plt.title('Stroke Rate by Smoking Status')

plt.tight_layout()
plt.show()
```

[57]

0 0 Indexing completed. Spaces: 4 LF Cell 8 of 40 Go Live


Search
EXPLORER
DMphase1.ipynb
NO FOLDER OPENED
OUTLINE
TIMELINE
MAVEN
PROJECTS
Data > tmp > documents > 1EB17CA6-2ABF-4D84-8568-E123638F8736 > DMphase1.ipynb
Talking:
Generate
Code
Markdown
Run All
Select Kernel
[57]
Python
Stroke Rate by Work Type
Stroke Rate
0.05
0.00
work_type_Self-employed
work_type_Private
work_type_children
work_type_Never_worked
Stroke Rate by Smoking Status
Stroke Rate
0.05
0.00
smoking_status_formerly smoked
smoking_status_smokes
smoking_status_never smoked
Python
0 0
Indexing completed.
Spaces: 4
LF
Cell 8 of 40
Go Live